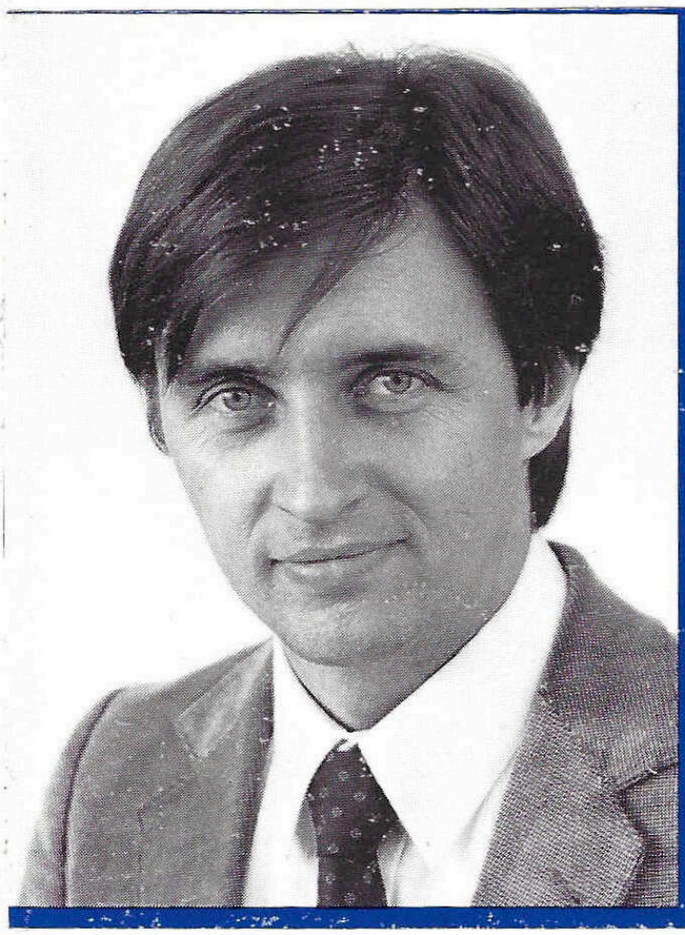
16 BIT GENERATION Z8000 AUFBAU UND ANWENDUNG

Peter Stuhlmüller

te-wi



Peter Stuhlmüller ist am 6. Januar 1951 in München geboren. Nach einem Studium der Nachrichtentechnik an der Ingenieurschule studierte er Regelungstechnik und Datenverarbeitung an der Technischen Universität München. Seine intensive Beschäftigung im Bereich der Elektronik ging Anfang 1977 in den Bereich der Mikrocomputertechnik über. Am Max-Planck-Institut für Biochemie nahe München begann für ihn im Zusammenhang mit einem spezifischen Projekt die Arbeit mit Mikrocomputern. Über Fachrecherchen baute sich 1979 die Neigung zu dem 16-bit-System Z 8000 auf, die er kompromißlos verfolgte. Das vorliegende Buch ist die logische Folge dieser intensiven Beschäftigung mit diesem Prozessor.

16 BIT GENERATION
Z8000
AUFBAU UND ANWENDUNG

Peter Stuhlmüller

16 BIT GENERATION Z8000 AUFBAU UND ANWENDUNG

Peter Stuhlmüller

te-wi
te-wi Verlag GmbH

IMPRESSUM

Verfasser: Peter Stuhlmüller

Alle Rechte vorbehalten. Ohne ausdrückliche, schriftliche Genehmigung der Herausgeber ist es nicht gestattet, das Buch oder Teile daraus in irgendeiner Form durch Fotokopie, Mikrofilm oder ein anderes Verfahren zu vervielfältigen oder zu verbreiten. Dasselbe gilt für das Recht der öffentlichen Wiedergabe.

©Copyright 1980 by te-wi Verlag GmbH

HERAUSGEBER:
te-wi Verlag GmbH, Waldfriedhofstraße 30
8000 München 70, Telefon 089/37 62 39

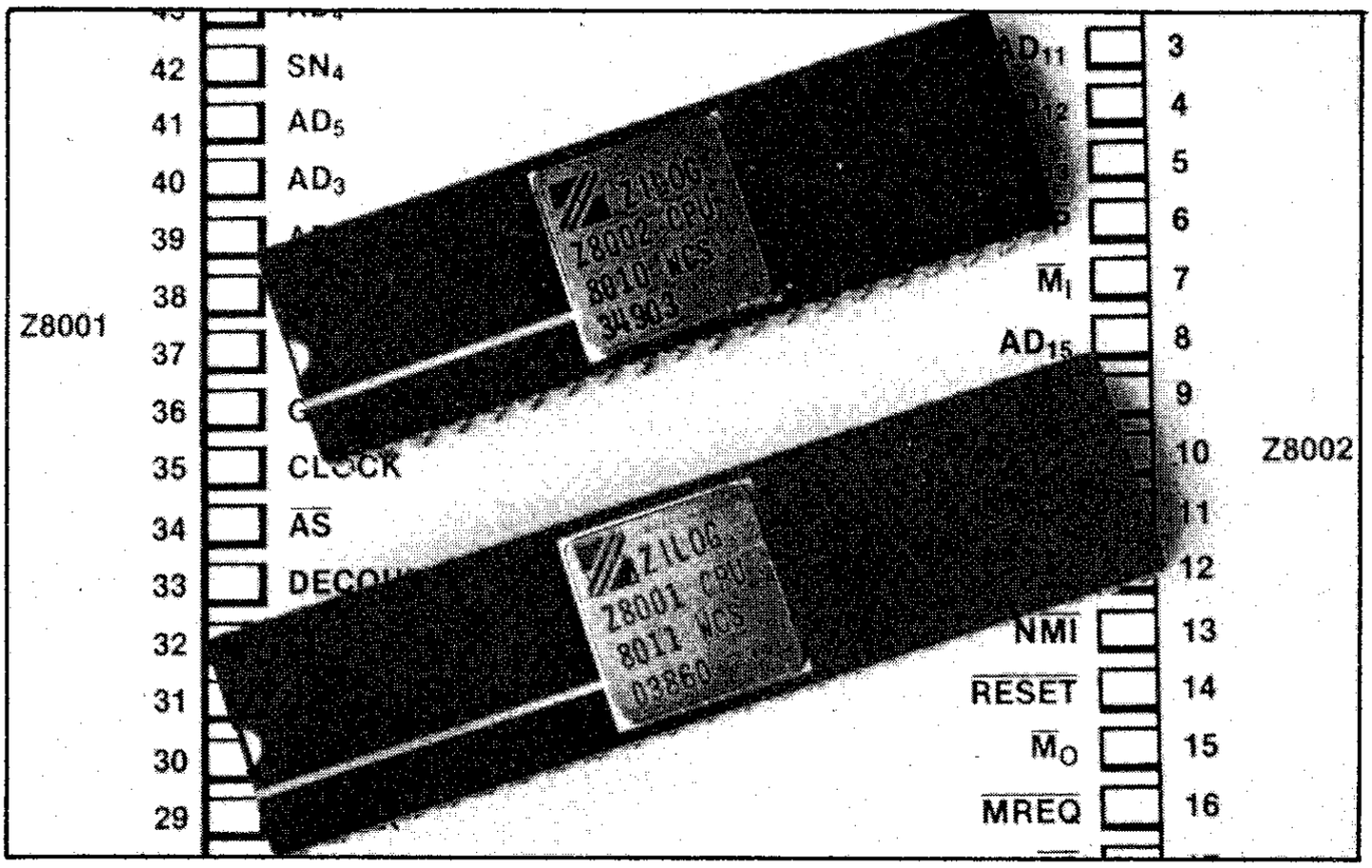
Die Herausgeber übernehmen keine Gewähr dafür, daß die beschriebenen Schaltungen, Baugruppen, Verfahren usw. funktionsfähig und frei von Schutzrechten Dritter sind.

GESAMTHERSTELLUNG:
technik marketing, München
tm2705/1280 1.Auflage
Printed in W-Germany

ISBN-3-921803-07-1

"DAS JAHRZEHNT DER SECHZEHNER"

"Wir stehen am Beginn des Zeitalters der Mikroelektronik" . . ., "Der Mikroprozessor hat seinen Siegeszug angetreten" . . ., "Der Mikro verändert unsere Umwelt" . . . So liest und hört man gleichsam mit Wohl- und Unbehagen. Wo stehen wir denn wirklich?



Wir stehen an der Schnittstelle zwischen fast einem Jahrzehnt der Entwicklung der Mikroelektronik und der Vision einer rechnergesteuerten, automatisierten Umwelt, in der der Begriff "Terminal" (Bildschirmarbeitsplatz) ein gängigerer Begriff wird, als beispielsweise "Blumenbeet im Garten". Sicherlich ist der Übergang in diese Zukunft kontinuierlich, aber doch sehr stetig. Wie negativ man die künftige Entwicklung auch sehen mag, für diejenigen, für die der Mikroprozessor im täglichen Leben einen beträchtlichen Teil ihres Denkens und Handelns bewirkt, ist sie keineswegs so destruktiv und kreativitätszerstörend, wie so manche Leute behaupten. Für die ständig steigende Zahl von Menschen, die den Mikroprozessor wirklich kennen, löst dieses winzige Ding immer wieder neue Faszination aus. Und an diese Kenner richtet sich auch dieses Buch.

Trotzdem, der Umgang mit µPs will gelernt sein, und man hat als Anwender auf praktisch-technischer Ebene nicht immer seine reine Freude mit den gut etablierten Typen 8080, 6800, SC/MP und vielen anderen. Zielt man hier auf die etabliert gemeinten 8-bit-Typen ab, so ist die nicht reine Freude darin zu begründen, daß die noch überwiegend große Maschinen- und Architekturabhängigkeit des Befehlssatzes eine Einengung bei der Programmierung auf Assemblerebene zur Folge hat.

Unter der erwähnten Schnittstelle ist im Detail eine weitere Entwicklung zu nennen. Die allseits bekannte, geradezu aggressive Entwicklung der Computer-Baustein-Technologie zeigt heute im Trend, wo es langgeht. Der 16-bit-Prozessor ist schon Realität. Diese Bausteine höchster Integrationsdichte, man nennt sie VLSI-Bausteine (very large scale integrated devices), sind mit weit mehr als nur dem doppelt so großen Datenformat wie 8-bit-Typen ausgestattet. Einer von den Herstellern oft und

gern posaunten, neuen Generation von Einchip-Prozessoren muß man hier die Berechtigung zuerkennen. Sowohl in der Architektur der Zentraleinheit (ALU, Register, Steuerlogik, Stacks, Status, Flags . . .) als auch im Befehlssatz wurden hier in der Tat beträchtliche Fortschritte gegenüber bisherigen 8-bit-Typen gemacht.

Ein sehr unwissenschaftlicher, aber doch anschaulicher Vergleich mit einem bekannten Utensil, dem Auto, drückt die Differenz etwa so aus: Die Hard- und Software-Charakteristiken zwischen 16- und 8-bit-Prozessoren entsprechen gleichsam den Fahrleistungen eines 700 kW-Turbo-Rennwagens gegenüber einem gängigen Mittelklassewagen mit 70 kW Leistung. Kenner wissen, welche Typen von den Rennern gemeint sind: Porsche 935 Turbo, BMW M1 Turbo oder Renault F1 Turbo. Es ist wohl kaum zu bestreiten, daß diese hochkarätigen Fahrzeuge mit ihren relativ niedervolumigen, aber technisch außerordentlich raffinierten Höchstleistungsmotoren bei jedem (nur einigermaßenen)Kenner eine echte Faszination auslösen.Ein in gleicher Weise außerordentlich aggressiver Vertreter der neuen 16-bit-CPU-Generation ist der Z 8000 der Firma Zilog, der gemeinsam von Zilog und AMD (Advanced Micro Devices) hergestellt wird.Für den Z 8000 bzw.AmZ 8000 gilt dieser Vergleich im wahrsten Sinne des Wortes.
Der Z 8000-Prozessor löst in vergleichbarer Weise auch die Faszination des Kenners im Computerbereich wie auch die des Kenners von Turbo-Rennwagen aus.
Dieses Buch richtet sich an all diejenigen Anwender von Mikrocomputern, die sich künftig diesem Superding hinwenden wollen.
Die 80er Jahre werden das Jahrzehnt der 16-bit-Prozessoren. Man sollte frühzeitig mit dem richtigen beginnen.

In eigener Sache:

Diese Dokumentation über den Z 8000 entstand unter erheblichem persönlichen Einsatz des Autors. Es sollte jedoch hierzu insbesondere bemerkt werden, daß dies ausschließlich aus eigenem Antrieb erfolgte. Die Idee dazu entstammte aus Detail-Fachrecherchen, die in die Zeit seiner Tätigkeit als Fachredakteur zurückreichten. Die daraus entstehende Begeisterung über diesen außerordentlichen Computerbaustein Z 8000 war letztlich die entscheidend treibende Kraft, die dieses Buch entstehen ließ. Nichtsdestoweniger sind alle Dinge im Leben, die nicht aus Zwang entstehen, die einzig wertvollen.
Der Autor wünscht aus Überzeugung, daß dem Leser dieses Buches die Begeisterung für den Z 8000 reproduzierbar erscheint, weil ihm die Größenordnung dieses Prozessors ersichtlich wird
Nicht zuletzt sollte an dieser Stelle den beiden Firmen Zilog und AMD für die Unterstützung ausdrücklicher Dank ausgesprochen werden, ohne dessen dokumentarische und technische Mittel dieses Buch nicht hätte entstehen können. Für die Firma Zilog ist Herr Dipl.-Ing. M. Scheurer (Regional Manager, Central Europe) und Herr Dipl.-Ing. W. Heer (Applikations-Abteilung) zu nennen.
Für Die Firma AMD ist Herr Ing. (grad.) D. Streit (Leiter der Applikation in Deutschland) genannt.
Ebenso ist auch den Mitarbeitern der Agentur tm für ihre rasche und präzise technische Abwicklung zu danken.

INHALTSVERZEICHNIS

Kapitel		Seite
1	**Einführung**	1-1
1.1	Was ist ein 16-bit-Computer?	1-1
1.2	Entwicklungsgeschichte und Strukturwandel	1-6
1.3	Einchip-16-bit-Prozessor oder Mehrchip-16-bit-Prozessorsystem: Eine Streitfrage?	1-12
2	**Die Einchip-16-bit-Zentraleinheit Z 8000**	2-1
2.1	Ein wenig Entwicklungsgeschichte	2-1
2.2	Technologie und Architektur einer Random orientierten CPU	2-4
2.3	Segmentierte und unsegmentierte Version	2-12
2.4	Anschlußbilder und Pinbelegung beider Versionen	2-17
2.5	Registeraufbau und -funktionen	2-23
2.6	Registeraufbau anderer 16-bit-Zentraleinheiten	2-39
2.7	Betriebszustandsanzeige während CPU-Operationen	2-50
2.8	Hardwarefunktionen des Z 8000-Bausteins, Signaldiagramme bei unterschiedlichen Betriebszuständen	2-56
2.9	Interruptbehandlung	2-70
2.10	Busanforderung von externer Operationseinheit (DMA)	2-83
2.11	Multiprozessorfähigkeit (hard- und softwaremäßig)	2-87
2.11.1	Multi-Mikro-Daisy-Chain-Buskonfiguration	2-92
3	**Adressierungsarten und Befehlssatz**	3-1
3.1	Die Adressierungsarten der Z 8000-CPU	3-2
3.2	Allgemeine Befehlsarchitektur	3-14
	Tabelle der Z 8000-Befehlsätze in alphabethischer Reihenfolge	3-16
3.3	Assembler, Befehlselemente, Symbole und Operationszeichen	3-25
3.4	Z 8000-Befehlsgruppen	3-31
3.4.1	Laden und Ändern von Register- und Speicherinhalten	3-31
3.4.2	Arithmetische Operationen	3-33
3.4.3	Logische Operationen	3-36
3.4.4	Programmsteueroperationen	3-38
3.4.5	Operationen für Einzelbit-Manipulation	3-41
3.4.6	Schiebe- und Rotationsoperationen	3-43
3.4.7	Block-Transfer und String-Operationen	3-45
3.4.8	Input/Output-Operationen	3-48
3.4.9	CPU-Steueroperationen	3-50
4	**Hardware-Konfigurationen für beide Z 8000-CPU-Typen**	4-1
4.1	Grundschaltung zur Bustrennung, Betriebszustands-Definition, Takterzeugung, Steuersignalpufferung und E/A-Adreßseparierung	4-1

Kapitel		Seite
4.2	Anschlußschematik für statische Schreib-/Lesespeicher	4-9
4.2.1	Beispiel eines Speicherwerks mit den RAM-Bausteinen 2114 bzw. Am 9124	4-14
4.2.2	Beispiel eines Speicherwerks mit dem Typ MK 4118 (1K x 8 bit)	4-14
4.2.3	Beispiel eines E/A-Speicherwerks mit dem Typ Am 9244 (4K x 1 Bit)	4-19
4.2.4	Rein statischer Speichertyp MK 4802 (2K x 8 Bit)	4-22
4.3	Anschlußschematik für quasi-statische Schreib-/ Lesespeicher	4-24
4.3.1	Was sind quasi-statische RAMs?	4-24
4.3.2	Beispiele für den Busanschluß mit den Typen MK 4816 und Z 6132	4-25
4.4	Anschlußschematik für dynamische Schreib-/Lesespeicher	4-38
4.4.1	Interface für Spalten-/Zeilen-Adreßmultiplex	4-45
4.4.2	Beispiel einer 64 KB Speicherbank mit 16K x 1 dynamischen RAMs	4-52
4.4.3	Das 32K x 1 bit dynamische RAM	4-71
4.4.4	Beispiel einer 64K Worte Speicherbank mit 64K x 1 bit dynamischen RAMs	4-73
4.5	Anschlußschematik für Festwertspeicher	4-76
4.5.1	Unterschiede bei der Anwendung von EPROMs und bipolaren PROMs	4-80
4.6	Anschlußschematik für gängige E/A-Bausteine mit 8-bit-Bus	4-82
4.7	Anschlußschematik für den System-Zeitgenerator-Baustein AmZ 8073	4-91
4.8	Fehlerkorrektur beim Betrieb dynamischer RAM-Speicherbänke	4-97
4.8.1	Beispiel eines dynamischen RAM-Speichersystems mit dem EDC 8160	4-101
4.9	Topologie und Konzeption eines kleinen Z 8002-Systems	4-104
5	**Der Speicherorganisations-Baustein Z 8010 MMU, ein Kernbaustein des Z 8000-Systems**	5-1
5.1	Prinzipielle Funktion der Z 8010-MMU	5-2
5.1.1	MMU-Register und Initialisierung	5-9
5.1.2	Definition des physikalischen Adreßfeldes	5-19
5.1.3	Signalverhalten der MMU	5-29
5.2	Anschlußbild der MMU	5-34
5.2.1	Logik für Schreibschutz des physikalischen Speicherfeldes	5-39
5.3	Anschlußschematik der Z 8010-MMU mit der Z 8001-CPU	5-40
5.3.1	Ein MMU-Baustein im Verbund mit der Z 8001-CPU	5-41
5.3.2	Zwei MMU-Bausteine im Verbund mit der CPU	5-42

Kapitel		Seite
6	**Assembler-Programmiertechnik mit den Z 8000-Befehlen**	6-1
6.1	Elementare Operationen (laden, rechnen, schieben, transferieren, Segmentsprung)	6-1
6.2	Verifikation einzelner Operationen	6-32
6.3	Ein Denkspiel programmiert, das Solitair-Spiel	6-34
6.4	Ein mathematisches Beispiel: schnelle Fourier-Transformation	6-44
6.5	Beispiel einer analogen Datenerfassung mit einem 16-bit-Analog-Digital-Umsetzer	6-55
7	**Ein Multiprozessor-Konzept mit Z 8000-Bausteinen**	7-1
7.1	Parallelrechnersysteme allgemein	7-3
7.2	Z 8000-Host- und Subsystem	7-7
7.2.1	Datenaustausch-Mechanismus	7-9
7.3	Hardware eines Z 8000-Parallelrechners nach "Master-Slave"-Prinzip	7-12
7.3.1	Multibus-System	7-12
7.3.2	Komponenten des Rechners	7-15
7.3.3	Steuerung durch Software mit Hardwareunterstützung	7-27
8	**Z 8000 – und wer noch**	8-1
8.1	Z 8000 – 68000, eine kleine Analyse	8-3
9	**Bestehende Z 8000-Systeme**	9-1
9.1	Z 8000 – Entwicklungs-Modul von Zilog	9-1
9.2	Evaluation-Board Am 96/4016 von AMD	9-9
10	**Das Z 8000-System für die 80er Jahre**	10-1
10.1	Systembausteine für allgemeine und spezifische Anwendungen	10-1
10.2	Rechnerkonzept mit Entlastungsprozessoren, die Z 8000-EPU	10-11
	ANHANG	
	Fachwörterverzeichnis	A

KURZVERZEICHNIS DER KAPITELBILDER

Bild-Nr.	Bezeichnung	Seite
1.2	Elementarer Aufbau eines Slice-Prozessorsystems	1-13
2.5	Gliederung der internen Logik des Z 8000	2-11
2.6	Fotografie des Z 8000-Chip	2-13
2.7	Gliederung der Bausteinanschlüsse, Z 8001/2	2-14
2.11	Anschlußbild der Z 8001-CPU	2-18
2.12	Anschlußbild der Z 8002-CPU	2-19
2.15	Flußdiagramm eines Benchmark-Programms	2-23
2.16	Operationen im Registerfeld bei der Programmausführung	2-25
2.17	Registerfeld der Z 8001-CPU	2-26
2.18	Registerfeld der Z 8002-CPU	2-27
2.22	Programm-Sprungtabelle beider CPUs	2-35
2.24	Funktion des Z 8000-Auffrischregisters	2-38
2.26	Registerfeld der 8086-CPU	2-43
2.28	Registerfeld der 68000-CPU	2-45
2.29	Registerfeld der LSI 11/23	2-46
2.30	Registerfeld der IBM 370	2-49
2.32	Signaldiagramm eines Z 8000-Holvorgangs	2-56
2.33	Phasen eines Befehlszyklus der Z 8000-CPU	2-59
2.34	Signaldiagramm einer Z 8000-Lese-/Schreiboperation	2-61
2.35	Signaldiagramm einer Z 8000-E/A-Operation	2-62
2.38	Signaldiagramm eines Int./Trap-Anerkennungszyklus	2-66
2.45	Signaldiagramm eines internen CPU-Zyklus	2-70
2.46	Verdrahtung einer priorisierten Interruptkette	2-76
2.51	Verdrahtung DMA-fähiger Peripheriebausteine	2-85
2.53	Z 8000 Multiprozessor-Konfiguration (Daisy Chain)	2-90
2.55	Z 8000 Multiprozessorsystem, allgemeines System	2-91
2.60	Flußdiagramm einer Multi-Mikro-Anforderungsroutine	2-97
3.1	Z 8000 Betriebsart und Speicherfeldaufteilung	3-3
3.2	Allgemeines Befehlsformat (häufige Form)	3-3
3.3	Direkte Registeradressierung	3-6
3.4	Unmittelbare Adressierung	3-6
3.5	Indirekte Adressierung	3-7
3.6	Direkte Adressierung	3-8
3.8	Indizierte Adressierung	3-10
3.9	Relative Adressierung	3-10
3.10	Basis-Adressierung	3-11
3.11	Basis-indizierte Adressierung	3-12
3.12	Strukturierte Darstellung der Adressierungsarten (Beispiele)	3-13
3.13	Befehlsformat eines Bedingungsbefehls	3-28
4.1	Blockdiagramm einer Bustrennschaltung	4-2
4.3	Grundschaltung Basisinterface (Bustrennung)	4-5
4.6	Generelle Struktur eines Speicheranschlusses an die CPU	4-11
4.7	Anschlußlogik für ein statisches Speicherwerk (B/W-Zugriff) **Hinweis:** ungerader Block → unteres Byte	4-12
4.8	Byteblock mit 1 K x 4 stat. RAMs	4-15
4.9	Byteblock mit 1 K x 8 stat. RAMs	4-16
4.13	E/A-Speicherwerk (4 K Worte) für einen ADC-Anschluß	4-21
4.15	Anschlußbild des quasi-statischen Speichers MK 4816	4-26
4.18	Speicherwerk mit der Kapazität 2 K Worte (MK 4816-2)	4-30

Bild-Nr.	Bezeichnung	Seite
4.22	Anschlußbild des quasi-statischen Speichers Z 6132	4-34
4.24	Speicherwerk mit der Kapazität 4 K Worte (Z 6132) **Hinweis:** A1 und B2 des LS 157 liegen auf Masse.	4-36
4.29	Anschluß und Signalverhalten des dynamischen RAM 4116	4-43
4.31	Prinzipschaltung der dynamischen RAM-Ansteuerung	4-46
4.32	Dynamisches Speicherwerk mit LS 157 als Multiplexer	4-47
4.36	Dynamisches Speicherwerk mit 3242 als Multiplexer (64 K Byte)	4-53
4.40	Anschlußbild des Multiplexbausteins AmZ 8164 (AMD)	4-57
4.42	Dynamisches Speicherwerk mit AmZ 8164 als Multiplexer (64 KB)	4-59
4.44	Schaltung des Datenausgangs der dynamischen Speicherwerke	4-62
4.45	Variationen der Freigabelogik (Dateninterface zum Bus)	4-63
4.50	16 K x 1 dyn. RAM, unterschiedliche 5V-Typen	4-70
4.51	32 K x 1 dyn. RAM, MK 4332, Anschluß und interner Aufbau	4-72
4.52	64 K x 1 dyn. RAM, MK 4164, Anschlußbild	4-74
4.53	64 K Worte Speicherwerk mit AmZ 8164 und MK 4164	4-75
4.54	Schaltung für einen Anschluß von Festwertspeichern	4-77
4.56	Anschlußbilder gängiger EPROMs (2 KB . . . 8 KB)	4-80
4.57	Anschlußbilder gängiger bipolarer PROMs	4-81
4.59	E/A-Schnittstelle mit 8 bit Einadreß-Ports 8212	4-83
4.60	E/A-Schnittstelle mit Dreifach-Port 9555	4-86
4.64	E/A-Schnittstelle mit der Z 80A PIO	4-90
4.65	Anschlußbild des Zählerbausteins AmZ 8073 STC	4-92
4.68	Schaltung eines Interfaces von CPU zu AmZ 8073	4-96
4.69	Anschlußbild des Fehlerkorrektur-Bausteins AmZ 8160 EDC	4-102
4.71	Interface: Z-Datenbus-EDC-Speicherwerk	4-105
4.72	Schaltung eines einfachen Z 8002-Systems	4-107
4.73	Adreßfeld des einfachen Z 8002-Systems	4-108
5.1	Interner Aufbau der MMU (Speicherorganisations-Baustein)	5-4
5.2	Physikalische Adreßberechnung bei langem Offset	5-5
5.3	Unterschied zwischen langem und kurzem Offset	5-7
5.5	Segment-Beschreibungsregister und Steuerregister der MMU	5-13
5.6	Zugriff auf Segment-Beschreibungsregister	5-14
5.8	Statusregister der MMU	5-16
5.12	Ein mögliches physikalisches Adreßfeld, adressiert von 2 MMUs	5-23
5.13	Segmentzuordnung physikalisch adressierter Segmente	5-24
5.14	Mögliche Segmentdefinition für einen Anwender	5-25
5.15	Mögliche Segmentdefinition für zwei Anwender	5-26
5.18	Leseoperation der CPU über die Z 8010 MMU	5-30
5.20	CPU-Steuer-(Initialisier-)Operation von und zu MMU	5-33
5.22	Gliederung der Bausteinanschlüsse der MMU	5-37
5.25	Inhibitlogik für eine Schreibunterbrechung in den Speicher	5-40
5.26	Z 8001-CPU in Zusammenarbeit mit einer MMU	5-41
5.27	Z 8001-CPU in Zusammenarbeit mit zwei MMUs	5-43
6.1	Datentypen der Z 8000-CPU	6-2
6.2	Zeichenkettenverarbeitung der Z 8000-CPU	6-3
6.4	Diagramm des Additionsbefehls, Registeradressierung	6-6
6.6	Diagramm des Subtraktionsbefehls, unmittelbare Adr.	6-7
6.7	Diagramm der Multiplikation, Registeradressierung	6-8
6.10	Rotation eines Registerinhalts	6-15
6.13	Verschiebung eines Registerinhalts nach rechts (arithm.)	6-17
6.15	Dynamische Verschiebung (arithmetisch)	6-18

Bild-Nr.	Bezeichnung	Seite
6.17	Diagramm des LDIR-Befehls	6-21
6.18	Diagramm des PUSH-Befehls, indirekt, Register	6-22
6.21	Diagramm des POP-Befehls, indirekt, indirekt	6-24
6.22	Diagramm des LDM-Befehls	6-25
6.23	Diagramm des CALL-Befehls	6-26
6.24	Diagramm des RET-Befehls	6-28
6.32	Flußdiagramm für ein Programm des Solitaire-Spiels	6-39
6.36	Diskrete Fourier-Transformation DFT, Formeln	6-47
6.37	Abtastung einer beliebigen, zeitinvarianten Funktion	6-48
6.40	Flußdiagramm des FFT-Algorithmus mit Eingangsbitumkehr	6-52
6.43	Z 8002-System berechnet eine vollständige FFT (Struktur)	6-55
6.44	Anschlußbild des ADC 71 von Burr Brown	6-56
6.46	Schaltung eines ADC-Interfaces an das Speicherwerk nach Bild 4.13	6-58
7.3	Z 8000 Multiprozessorsystem (Master-Slave-Struktur)	7-13
7.4	Host-Computerteil des Multiprozessorsystems	7-14
7.5	Subsystem des Multiprozessorsystems	7-16
7.7	Hauptspeicherwerk des Multiprozessorsystems	7-19
7.11	Synchrones Interrupt-Netzwerk (Struktur)	7-24
7.12	Schaltung des Interrupt-Netzwerks	7-25
7.14	E/A-Schnittstellen-Netzwerk des Multiprozessorsystems	7-28
8.1	Fotografie des 68000-Chip	8-2
8.2	Gliederung der Bausteinanschlüsse der 68000-CPU	8-3
8.3	Anschlußbild der 68000-CPU	8-4
9.1	Z 8000-Einplatinencomputer (Entwicklungsmodul)	9-2
9.3	Systemverbindung: Z 8000 DM, PDS 8000, Drucker, Terminal	9-4
9.4	Emulatorsystem für beide CPUs: Zilog ZSCAN 8000	9-7
9.5	Systemverbindung: ZSCAN mit Emulatorleitung zu Z 8000-Anwenderhardware	9-8
9.6	Evaluation-Board AmZ 8000 des Typs AMC 96/4016	9-10
10.1	Zusammenstellung der Z-Bus-Signalleitungen	10-2
10.2	Grundschaltung für den Anschluß von Z 8000-Peripherie	10-3
10.3	Anschlußbild des Z 8030 SCC mit charakteristischem Symbol	10-5
10.4	Anschlußbild des Z 8034 UPC mit charakt. Blocksymbol	10-6
10.5	Anschlußbild des Z 8036 CIO mit charakt. Blocksymbol	10-7
10.6	Anschlußbild des Z 8038 FIO mit charakt. Blocksymbol	10-9
10.7	Charakteristisches Blocksymbol des FIO-Zusatzbausteins Z 8060	10-10
10.8	Prinzipieller Anschluß des FIO an die CPU	10-11
10.9	Kaskadierung eines FIO-Speichersystems	10-12
10.10	Anschlußbild des Z 8065 BEP mit charakt. Blocksymbol	10-13
10.12	Busverbindung der Z 8000-CPU mit der Z 8000-EPU	10-15
10.13	Struktur eines Z 8000-Systems mit Entlastungsprozessoren	10-17

Z 8000-BEFEHLSSATZ

Befehlsbeschreibung	Mnemonic	Daten-art	Adressierungsart	Flags aktiv
Arithmetische Befehle				
Add	ADD	B, W, L	R, IM, IR, DA, X	C, Z, S, V, D^1, H^1
Add with Carry	ADC	B, W	R	C, Z, S, V, D^1, H^1
Compare (Immediate)	CP	B, W	IR, DA, X	C, Z, S, V
Compare (Register)	CP	B, W, L	R, IM, IR, DA, X	C, Z, S, V
Decimal Adjust Bit	DAB	B	IR	C, Z, S
Decrement	DEC	B, W	R, IR, DA, X	Z, S, V
Divide	DIV	W, L	R, IM, IR, DA, X	C, Z, S, V
Extend Sign	EXTS	B, W, L	R	C, Z, S, V
Increment	INC	B, W	R, IR, DA, X	Z, S, V
Multiply	MULT	W, L	R, IM, IR, DA, X	C, Z, S, V
Negate	NEG	B, W	R, IR, DA, X	C, Z, S, V
Subtract	SUB	B, W	R	C, Z, S, V, D^1, H^1
Subtract with Carry	SBC	B, W, L	R, IM, IR, DA, X	C, Z, S, V, D^1, H^1
Bit-Manipulation				
Bit Test	BIT	B, W	R	Z
Bit Reset (Static)	RES	B, W	R, IR, DA, X	–
Bit Reset (Dynamic)	RES	B, W	R	–
Bit Set (Static)	SET	B, W	R, IR, DA, X	–
Bit Set (Dynamic)	SET	B, W	R	–
Bit Test and Set	TSET	B, W	R, IR, DA, X	S
Block-Transfer und String-Manipulation				
Compare and Decrement	CPD	B, W	IR	C, Z, S, V
Compare, Decrement and Repeat	CPDR	B, W	IR	C, Z, S, V
Compare and Increment	CPI	B, W	IR	C, Z, S, V
Compare, Increment and Repeat	CPIR	B, W	IR	C, Z, S, V
Compare String and Decrement	CPSD	B, W	IR	C, Z, S, V
Compare String, Decrement and Repeat	CPSDR	B, W	IR	C, Z, S, V
Compare String and Increment	CPSI	B, W	IR	C, Z, S, V
Compare String, Increment and Repeat	CPSIR	B, W	IR	C, Z, S, V
Load and Decrement	LDD	B, W	IR	V
Load, Decrement and Repeat	LDDR	B, W	IR	V
Load and Increment	LDI	B, W	IR	V
Load Increment, and Repeat	LDIR	B, W	IR	V
Translate and Decrement	TRDB	B	IR	Z, V
Translate, Decrement and Repeat	TRDRB	B	IR	Z, V
Translate and Increment	TRIB	B	IR	Z, V
Translate, Increment and Repeat	TRIRB	B	IR	Z, V
Translate, Test, and Decrement	TRTDB	B	IR	Z, V
Translate, Test, Decrement, Repeat	TRIDRB	B	IR	Z, V
Translate, Test, and Increment	TRTIB	B	IR	Z, V
Translate, Test, Increment and Repeat	TRTIRB	B	IR	Z, V

Befehlsbeschreibung	Mnemonic	Daten-art	Adressierungsart	Flags aktiv
CPU-Steueroperationen				
Complement Flag	COMFLG	–	–	$C^2, 2^2, S^2, P^2, V^2$
Disable Interrupt	DI	–	–	–
Enable Interrupt	EI	–	–	–
Halt	HALT	–	–	–
Load Control Register (from register)	LDCTL	–	R	$C^2, 2^2, S^2, P^2, D^2, H^2$
Load Control Register (to register)	LDCTL	–	–	–
Load Program Status	LDPS	–	IR, DA, X	C, Z, S, P, D, H
Multi-Bit Test	MBPT	–	–	S
Multi-Micro Request	MREQ	–	–	Z, S
Multi-Micro Reset	MRES	–	–	–
Multi-Micro Set	MSET	–	–	–
No Operation	NOP	–	–	–
Reset Flag	RESFLG	–	–	$C^2, 2^2, S^2, P^2, V^2$
Set Flag	SETFLG	–	–	$C^2, 2^2, S^2, P^2, V^2$

B = Byte, W = Wort, L = Langwort

1) Flags nur bei Byte-Operationen aktiv
2) Änderung der Flags bei spezifischer Befehlsausführung

Befehlsbeschreibung	Mnemonic	Daten-art	Adressierungsart		Flags aktiv
Input/Output-Befehle[3]			**üblich**	**speziell**	
Input	(S)IN	B, W	IR, DA	(DA)	–
Input and Decrement	(S)IND	B, W	IR	(IR)	V
Input, Decrement and Repeat	(S)INDR	B, W	IR	(IR)	V
Input and Increment	(S)INI	B, W	IR	(IR)	V
Input, Increment and Repeat	(S)INIR	B, W	IR	(IR)	V
Output	(S)OUT	B, W	IR, DA	(DA)	–
Output and Decrement	(S)OUTD	B, W	IR	(IR)	V
Output, Decrement, and Repeat	(S)OUTDR	B, W	IR	(IR)	V
Output and Increment	(S)OUTI	B, W	IR	(IR)	V
Output, Increment, and Repeat	(S)OUTIR	B, W	IR	(IR)	V
Logical Instructions					
And	AND	B, W	R, IM, IR, DA, X		Z, S, P
Complement	COM	B, W	R, IR, DA, X		Z, S, P
Or	OR	B, W	R, IM, IR, DA, X		Z, S, P
Test	TEST	B, W, L	R, IR, DA, X		Z, S, P
Test Condition Code	TCC	B, W	R		–
Exclusive Or	XOR	B, W	R, IM, IR, DA, X		Z, S, P
Program Control Instructions					
Call Procedure	CALL	–	IR, DA, X		–
Call Procedure Relative	CALR	–	RA		–
Decrement, Jump if Not Zero	DINZ	B, W	RA		–
Interrupt Return	IRET	–	–		C, Z, S, P, D, H
Jump	JP	–	IR, DA, X		–
Jump Relative	JR	–	RA		–
Return from Procedure	RET	–	–		–
System Call	SC	–	–		–
Rotate and Shift Instructions					
Rotate Left	RL	B, W	R		–
Rotate Left Through Carry	RLC	B, W	R		C, Z, S, V
Rotate Left Digit	RLDS	B	R		Z, S
Rotate Right	RR	B, W	R		C, Z, S, V
Rotate Right Through Carry	RRC	B, W	R		C, Z, S, V
Rotate Right Digit	RRDB	B	R		Z, S
Shift Dynamic Arithmetic	SDA	B, W, L	R		C, Z, S, V
Shift Dynamic Logical	SDL	B, W, L	R		C, Z, S, V
Shift Left Arithmetic	SLA	B, W, L	R		C, Z, S, V
Shift Left Logical	SLL	B, W, L	R		C, Z, S, V
Shift Right Arithmetic	SRA	B, W, L	R		C, Z, S, V
Shift Right Logical	SRL	B, W, L	R		C, Z, S, V

B = Byte, W = Wort, L = Langwort

3) Jeder I/O-Befehl (ausgenommen (S)IN und (S)OUT) bezeichnet sein eigenes Zählerregister, das die Anzahl der transferierten Operanden beinhaltet.

Einführung

1

1.1 WAS IST EIN 16-BIT-COMPUTER?

"Dumme Frage. Ein 16-bit-Computer beinhaltet eine 16-bit-Zentraleinheit, die sich durch die Breite des Datenformates charakterisiert, eben die 16 bit. Und was den Aufbau des Computers betrifft: RAM, ROM und Schnittstellen gehören dazu. Das braucht jeder Computer. Also kurzum, im Prinzip dasselbe wie bei einem 8-bit-Computer . . ."

So könnte man auf die Frage kurz und bündig antworten. Die obige Argumentation ist sicherlich richtig. In der Kürze liegt die Würze und je kürzer man antwortet, desto weniger ist daran falsch (bekanntlich). Wenn es doch nur so einfach wäre, dieses Buch wäre in der Tat überflüssig. Einen 16-bit-Computer an Hand eines 8- oder gar 4-bit-Systems zu erklären, ist gewiß ein gangbarer Weg, aber für ein echtes 16-bit-System keineswegs empfehlenswert. Will man das Thema 16 bit wirklich ernst nehmen, wäre dieser Weg einfach nicht zeitgemäß. Wir befinden uns nicht mehr am Anfang der Computerentwicklung, obwohl sie erst vor einer Menschengeneration richtig begann. Man muß zwischen Systemen verschiedenen Datenformats schon ernsthaft differenzieren.

Computersysteme mit einer Zentraleinheit in Form eines zentnerschweren Schranks voller Elektronik plus weiteren Schränken für die Speichereinheiten haben vor etwa 15 Jahren — und mehr — den eigentlichen Typus Computer schlechthin repräsentiert. Ihre Architektur galt damals als richtungsweisend. Computer beanspruchten eben ihren Raum und es war zugleich auch ein gewisses äußeres Merkmal von Macht. Die Rede ist von der Computergeneration der dritten Art, die noch mit diskreten Transistoren, aber schon vereinzelt mit integrierten Schaltungen auf TTL-Basis aufgebaut waren. Diese Maschinen (im wahrsten Sinne des Wortes) waren in ihrem Aufbau so ausgelegt, daß man numerische Algorithmen algebraischer und infinitesimaler Art im Bereich großer Zahlenwerte berechnen konnte. Rechnen stand damals überwiegend im Vordergrund bei der Anwendung von Computern. Die Architektur der Zentraleinheit war bei diesen Rechnern noch keineswegs auf strukturierte Programmierung in höheren Sprachen ausgelegt. Das Rechen- und Steuerwerk und das beschränkte Registerfeld besaßen den damaligen Möglichkeiten entsprechend eine klassische Konzeption. Echte online-Prozeßanwendungen waren Mitte der 60er Jahre noch so gut wie unbekannt.

Gegen Ende der 60er Jahre ergab sich dann eine grundsätzliche Trennung der Rechnerentwicklung. Zwei Richtungen entstanden. Eine der beiden war von der Fortführung der klassischen Art der Computerentwicklung geprägt. Sie vollzog sich in Richtung zur Großrechnerarchitektur mit dem dieser Art charakteristischen Datenformat von 32, 48 und 64 bit. Diese mit außerordentlich aufwendiger Technik konzipierten Rechner enthielten diskrete Spezialtransistoren und SSI-Schaltungen (vorwiegend in Schottky- und ECL-Technologie) geringer Dichte, aber mit Schaltzeiten im Sub-Nanosekunden-Bereich. Diese Rechner repräsentierten dann das obere Ende der Größenordnung von Computern. Sie gibt es auch heute noch.

Die zweite Richtung der Entwicklung von Computern vollzog sich ausgespro-

chen nichtklassisch. Computerschränke beträchtlicher quantitativer Ausmaße konnten einer breiten Anwendung in Industrie und Handel nicht hilfreich sein, weil die immensen Hardwarekosten eine Verbreitung in großem Maßstab verhinderten. Um eine Verbreitung in größerem Maßstab und vielfältige kommerzielle Anwendungen zu erreichen, mußte die Computerindustrie den Computer in kleinen Ausmaßen und zu einem geringen Preis anbieten. Die Verdichtung der Integration von Transistoren zu einer integrierten Logikschaltung war der einzig gangbare Weg zu einer gut überschaubaren Computerarchitektur. Der massive Einsatz von Mitteln trug schon relativ früh (1970/71) zu ersten Erfolgen bei. Die 4-bit-CPU des Typs 4004 von der Firma Intel und die erste 8-bit-CPU der gleichen Firma mit der Bezeichnung 8008 erweckte in der Tat Hoffnungen für die Zukunft. Mit dem Erscheinen des 8080, der verbesserten Version des 8008, im Jahre 1973 hatte die Firma Intel wohl den Startschuß in das Zeitalter der Mikroelektronik gegeben. Eine außerordentlich rasante Entwicklung begann. Die späteren 70er Jahre gehörten dem 8-bit-Prozessor, dessen verbesserte Varianten auch in den 80er Jahren noch sehr hoch im Kurs stehen werden. Der vielfache Einsatz der 8-bit-Prozessoren in Tischrechnern (verifiziert auch als Personal-Computer bezeichnet), Steuercomputern in technischen Geräten (on-line) oder in mehrfacher Ausführung in dezentral konzipierten Datenverarbeitungssystemen brachte ein weites Spektrum an Erfahrungen mit sich. Darüberhinaus ergaben diese Anwendungen den Nachweis, daß in zahlreichen Fällen das 8-bit-Format reicht, wenn die zeitliche Sequenz der Aufgabe dies zuläßt.

Eine grundsätzliche Erfahrung beim Einchip-Prozessor/Computer zeigte sich jedoch in eindrucksvoller Weise. Ein relativ kleines 8-bit-Mikrocomputersystem kann zuweilen Anforderungen erfüllen, die noch vor nicht all zu langer Zeit einem diskreten 16-bit-System übertragen, oder in einer aufwendigen, festverdrahteten Logik realisiert wurde. Die festverdrahtete Logik war lange Zeit im kommerziellen Einsatz priviligiert. Heute weiß man, daß ein kleines μP-System beträchtliche Kosten sparen kann. Besonders in on-line-Anwendungen hat sich der Mikrocomputer einen festen Einsatzbereich gesichert. Wenn sich auch mit 8-bit-Systemen viele Aufgaben lösen ließen, bleiben doch kommerzielle Bereiche ab einem bestimmten Niveau den schon anfangs genannten großen Kästen vorbehalten.

Die großen Kästen verbargen nun mal 16-bit-Zentraleinheiten oder größere. Dabei darf keinesfalls vergessen werden, daß diese Rechner auch an dem Boom der integrierten Bausteine eine Verbesserung ihrer Struktur erfuhren. Trotzdem, die Konsequenz der Mikrocomputerhersteller war zwangsläufig die, aufbauend auf dem Erfolg der 8-bit-Prozessoren, den Angriff auf die großen Kästen zu wagen, oder besser, die Herausforderung anzunehmen. Gelingt es, 16-bit-Prozessoren zu entwickeln, die die Eigenschaften von "diskreten" Zentraleinheiten in einem Chip vereinigen, so könnten damit künftige Rechnerarchitekturen entscheidend beeinflußt werden. Man könnte den attraktiven, kommerziellen 16-bit-Bereich abdecken und den Rechnerboom nochmals kräftig anheizen.
Computer mit einem 16-bit-Datenformat werden ziemlich pauschal als Minicomputer bezeichnet. Diese Pauschalierung wurde schon Mitte der 60er Jahre eingeführt. Woher kam das? Etwa Mitte der 60er Jahre zeichnete sich eine gewisse Spaltung in der Rechnerkonzeption ab. Einige ganz wenige Firmen der Computerbranche drifteten mit ihren Produkten deutlich nach oben hin ab, d.h. sie entwarfen echte "Riesen". Die Firma Control Data ist hier mit Nachdruck zu nennen. Nachdem der Großrechner CD 6000 im Jahre 1964 von dieser Firma vorgestellt wurde, war eine Qualifizierung nach dem Datenformat des Rechners nicht auszuschalten. Ein 16-

bit-System ist eben neben einem 64-bit-Monstrum, wie es diese Control Data-Rechner nun mal ansich haben, wirklich nur ein Zwerg. Die Bezeichnung Mini hat keine genau zu lokalisierenden Gründe. Ein Minicomputer kommt mit dieser Bezeichnung vom Hersteller selbst. Vielleicht hat dieses Prädikat "Mini" etwas mit den zu der damaligen Zeit aktuellen Mini-Röcken zu tun. Genau weiß man es nicht. Minicomputer klingt jedenfalls etwas diskriminierend, zumal man weiß, daß ihre Leistungsfähigkeit in den letzten Jahren noch beträchtlich zugenommen hat. Mikro dagegen steht für Mikroelektronik und klingt hier keineswegs abwertend.

Im Verlauf der 70er Jahre wurde für das 16-bit-Format etwas aufwertend das Prädikat "Mittlere Datentechnik" geprägt. Im weiteren Verlauf wurden 16-bit-Systeme geradezu hochgehievt, indem sie mit dem Bereich Basisdatenverarbeitung gleichgesetzt wurden. Festzustellen ist insgesamt dies: 16-bit-Systeme "diskreter" Art haben im Verlauf der 70er Jahre eine außerordentliche Leistungsfähigkeit erlangt. Ein Vergleich zu 8-bit-Systemen ist im Grunde nicht gerechtfertigt, weil hier von ungleichen Voraussetzungen ausgegangen wird.

Die Adressierungsbereiche dieser mit MSI- und LSI-Bausteinen aufgebauten Rechner reichen bis zu 16 MByte. Es wird in Segmenten adressiert, die mit Speicherschutzvorkehrungen zu sichern sind (schreiben und lesen). Bei den IBM-Systemen werden sogenannte Speicherschutzschlüssel verwendet. Es wird ferner mit virtuellen Speicherfeldern gearbeitet. Man versteht darunter ein externes Speichermedium (z.B. Fest-/Wechselplattenspeicher), das die tatsächliche physikalische Größe des Hauptspeichers ausbaut. Die daraus resultierende wirkungsvollere Ausnutzung des Hauptspeichers ermöglicht das Eingliedern sehr großer Programme (200 KB und größer) ohne Überlagerung in den Hauptspeicher. Die Zentraleinheit arbeitet mit virtuellen und wirklichen Programmadressen. In der virtuellen Arbeitsweise wird durch die dynamische Adreßumsetzung der Hauptspeicher in realen Adressen angesprochen. Programme im virtuellen Modus werden in Bereiche (Segmente oder Seiten) unterteilt und nur die unmittelbar zur Ausführung anstehenden Programmelemente in den Hauptspeicher geladen. Bei einem Bedarf von weiteren Programmelementen wird aus dem externen Medium entsprechend in den Hauptspeicher transferiert. Beim Arbeiten mit realen Speicheradressen sind virtuelle und reale Adressen identisch. Es ist keine Übersetzung segmentierter Programme notwendig. Das gesamte Programm befindet sich im Hauptspeicher.

Die dynamische Adreßumsetzung wird in ziemlich massiver Form bei größeren Systemen angewandt. Sie gehört zum übergeordneten Speichermanagement der Maschine.
Der Begriff "virtueller Speicher" wird nochmals im Kapitel 5 dieses Buches aufgegriffen.

Als Beispiel einer echten 16-bit-Hochleistungsmaschine sei die Eclipse C oder die Nova 3/D von der Firma Data General genannt. Die Speicherorganisations- und Schutzeinheit dieser Computersysteme setzt den logischen Adreßbereich in einen 128 K Worte umfassenden physikalischen Adreßfeldbereich um. Vier Adreßerweiterungstabellen (zwei für den Programmcode und zwei für Datenfelder) ergeben diese Feldstrukturierung. Weitere Eigenschaften wie priviligierte Befehle, spezielle Behandlung von E/A-Schnittstellen und ein Zugriffsschutz in Hauptspeicherabschnitten kennzeichnen diese modernen 16-bit-Systeme. Ein weiteres Beispiel der 16-bit-Leistungsklasse demonstrieren die Minicomputer des Typs 621 (X2(X3) der Firma Dietz. Diese auf bipolarer Technologie basierenden Systeme mit einer Maschinenzykluszeit von 70 ns sind ebenfalls mit einer übergeordneten Hauptspeicherorganisation ausgestattet. Die Einteilung des Hauptspeicherfeldes wird durch 32 Kachelregister koordiniert, die die dynamische Zuordnung von Programmebenen über ein dynami-

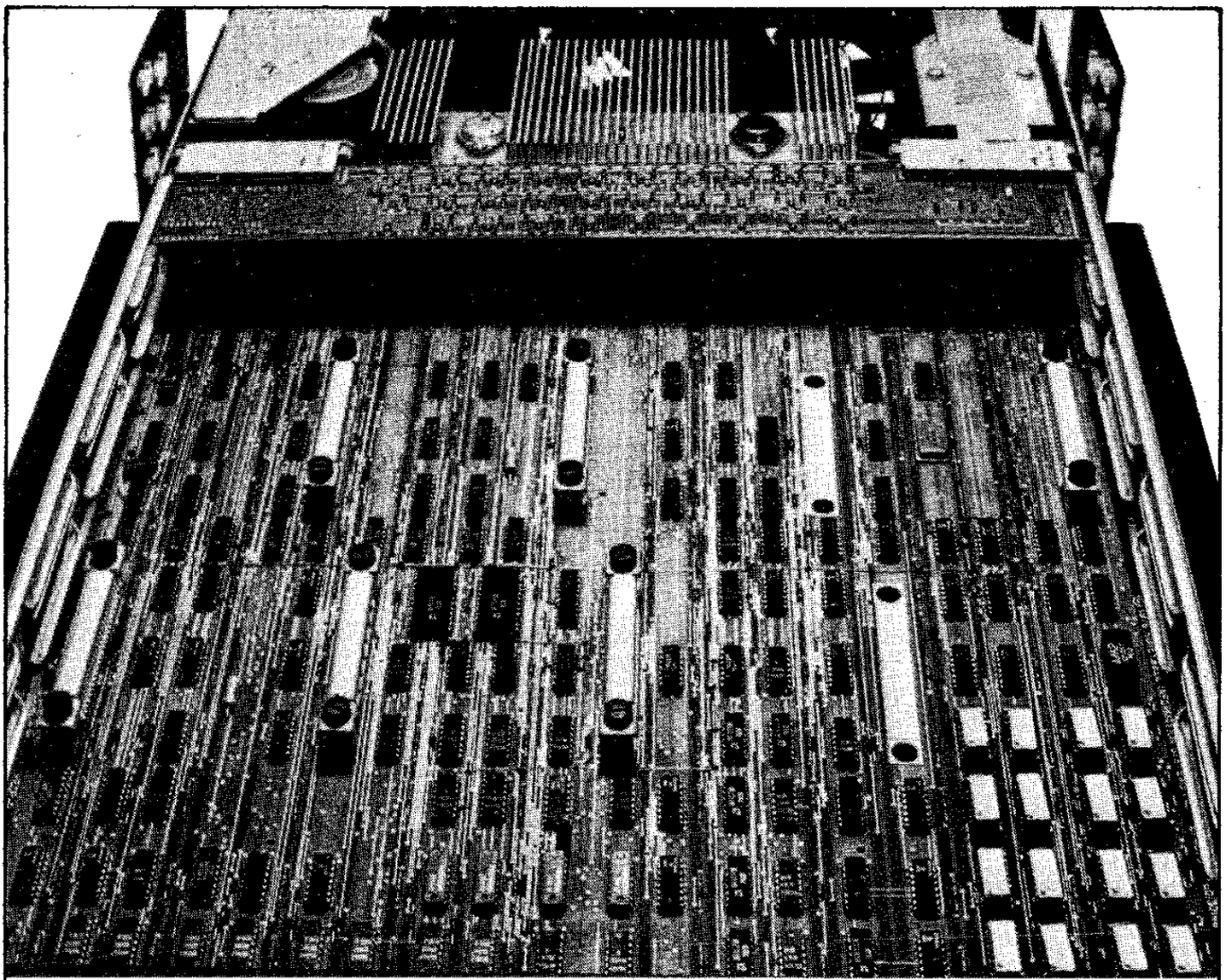
sches Adreßfeld von 1 MByte erreicht. Auch hier wird wieder in Programm- und Datenbereiche unterschieden. Man kann logisch zusammenhängende Benutzerbereiche definieren und vor der Überschreibung schützen.

Bei Systemen der genannten Größenordnung verfügt die Zentraleinheit zum Teil schon über zwei arithmetische Logikeinheiten, die z.B. einen Hardware-Multiplizierer enthalten. Parallel dazu wurde im Rahmen der ständig zunehmenden Aufgabenbereiche von 16-bit-Systemen die direkte Einsatzfähigkeit als Prozeßrechner wesentlich verbessert. Das heißt nichts anderes als daß die Interrupt-Strukturen verfeinert wurden und durch die Steigerung des Datendurchsatzes schnelle Echtzeitanwendungen möglich wurden. Der on-line-Einsatz von Rechnern, speziell des 16-bit-Bereichs, ist nun mal auf Grund der Automatisierungsbestrebungen der Industrie ganz beträchtlich gestiegen.

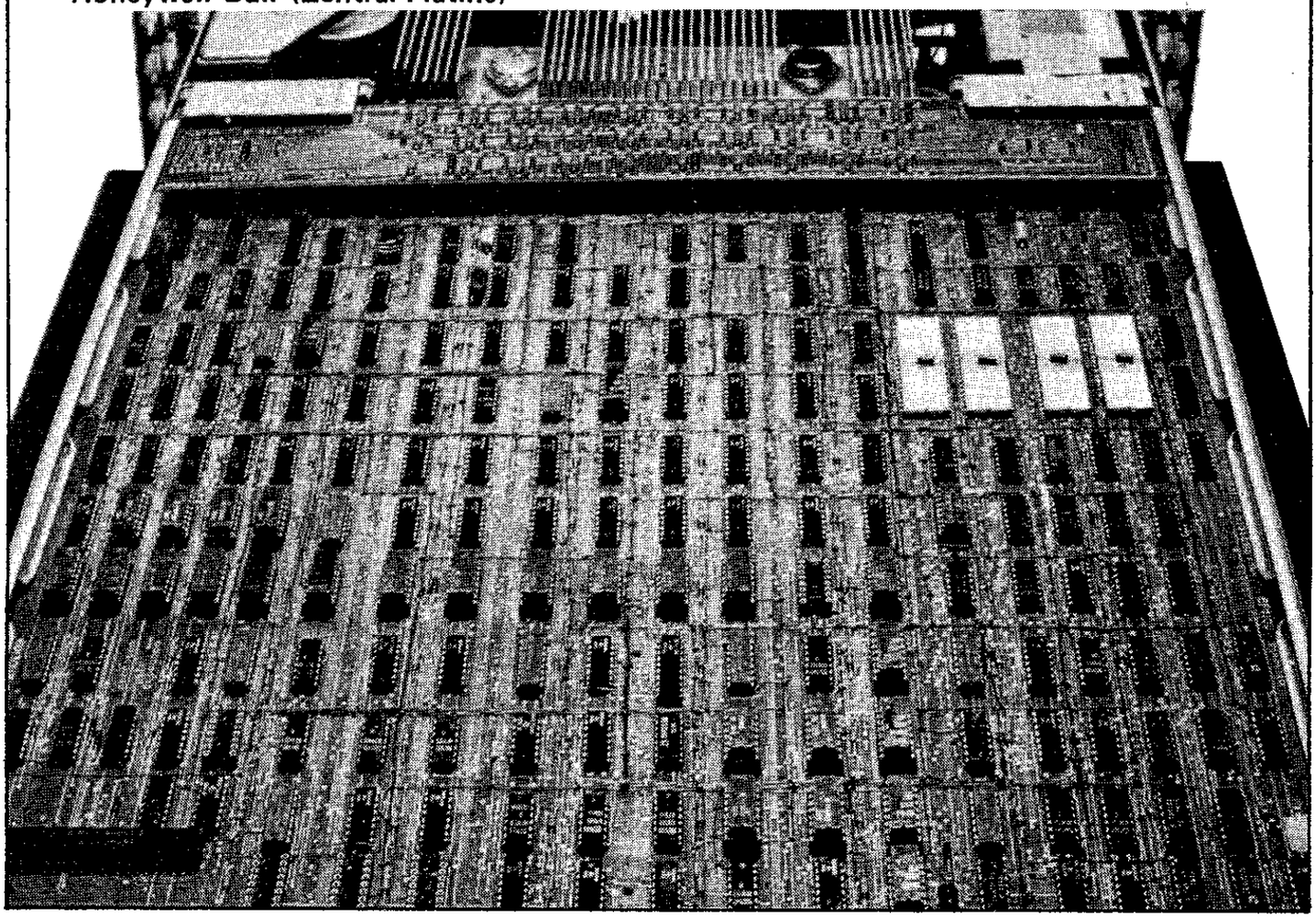
Das obere Ende der Skala im 16-bit-Bereich ist u.a. das System 6 der Serie 60 von Honeywell Bull, die bis zu 16 MByte (Modell 53) direkt adressieren können. Es beinhaltet einen Hardware-Gleitkomma- und COBOL-Prozessor als Entlastungsprozessor zur Zentraleinheit. Zur Verwaltung der Peripherie werden Schnittstellenprozessoren, Magnetplatten- und Magnetbandprozessoren eingesetzt. 26 Akkumulator- und Indexregister zeigen bei dieser neueren Maschine den Trend zu hochregisterorientierten Architekturen auf. Die Interrupt-Struktur ist mit der automatischen Sicherung und Wiederherstellung von Registerinhalten gekoppelt, so daß auf jeder Programmebene die Voraussetzung einer ordentlichen, breiten Priorisierung der Interrupts gegeben ist. 64 gewichtete Interrupts sind möglich. Das Hauptspeicherfeld, bestehend aus 16 K x 1 dynamischen RAMs (Schreib-/Lesespeicher), hat für die MByte-Speicherfelder auch eine Hardware-Fehlererkennungs- und Korrekturlogik mit inbegriffen. Beim Lesen des 22-bit-Wortes werden 1-bit-Felder automatisch korrigiert und Mehrbit-Fehler erkannt und gemeldet. Über Speichersegmentbereich und Schutzvorrichtungen wurden schon Andeutungen gemacht. Das System 6 ist hier mit einigen Raffinessen ausgestattet. Was die Adreßsegmente und den Schreibschutz betrifft, werden in diesem Buch noch an anderer Stelle Einzelheiten dargestellt. Was das System 6 von Honeywell Bull angeht: hier treten bereits Grenzbereiche auf. Ein System mit verteilter Intelligenz in Form von Entlastungsprozessoren neben dem eigentlichen Zentralprozessor, welch einen Namen verdient dieses System? Multiprozessor-Rechner ...? Es wären hier noch einzelne 16-bit-Systeme der oberen Größenordnung und ihre Raffinessen zu beschreiben, um das Spektrum zu vervollständigen. Zum Ausdruck sollte hier im Grunde nur kommen, daß der 16-bit-Bereich wirklich eine eigene Computerwelt ist. Im Laufe der 70er Jahre hat sich hier eine enorme Computer-Power (beziehend auf die Hardware) in diesen großen Kästen etabliert. Die anfangs etwas diskriminierend klingende Beschreibung der zentnerschweren Rechnerschränke wurde damit einigermaßen zurechtgerückt. Im kommerziellen Einsatz stehen diese Systeme sehr hoch im Kurs. Ein Vergleich zu erwähnten 8-bit-Mikrocomputersystemen ist illusorisch, wenn nicht gar lächerlich.

Mit der kurzen, sicherlich unvollständigen, aber einigermaßen anschaulichen Darstellung des gegenwärtigen 16-bit-Bereichs (bezogen auf die "diskreten" Maschinen) sollte die anfänglich gemachte Bemerkung über die Differenzierung etwas transparenter verstanden werden. Einchip-16-bit-CPUs einerseits und "diskrete" Zentraleinheiten andererseits.

Will sich ein Einchip-16-bit-Prozessor und seine systemverwandten Peripheriebausteine wirklich auf dem Markt des mittleren Computerbereichs durchsetzen, so muß sich die Leistungsfähigkeit dieser 16-bit-Familie an der echten Computer-Power der genannten "diskreten" Systeme orientieren. Andernfalls ist ein Erfolg auf breiter Basis nicht ge-



Innenansicht des 16 bit Hochleistungs-Computersystems "System 6" der Serie 60 von Honeywell Bull (Zentral-Platine)



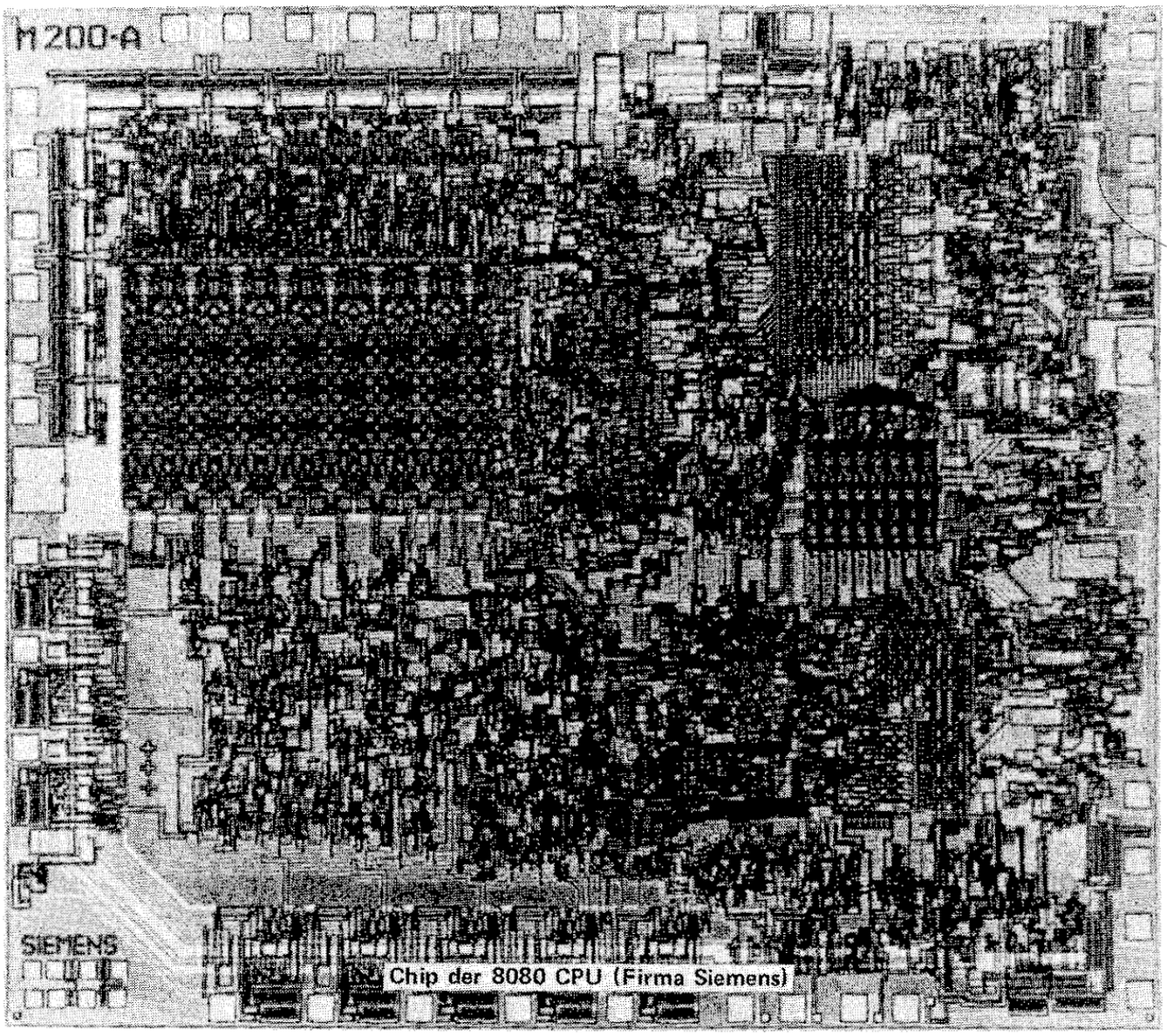
geben. Welche Konsequenz ist daraus weitgehend zu ziehen? Sollen komplette Platinen in den "diskreten" Systemen durch einen Chip ersetzt werden, sind an die Dichte dieser VLSI-Bausteine (very large scale integration devices) ausgesprochen extreme Anforderungen gestellt. 20.000 Transistoren auf einem Chip wird und ist bereits eine gängige Größenordnung. Rückwirkungen hat das bereits auf die Halbleiterprozesse genommen. Der Standard-NMOS-Prozeß (N-Kanal-Verarmungstyp-Siliziumprozeß), der den Basisherstellungsprozeß moderner 8-bit-CPUs darstellt, wurde bereits weiter verdichtet. Der bereits erwähnte Angriff der Computer-Bauelementehersteller in den 16-bit-Bereich beinhaltet selbstverständlich die gegebenen Beispiele der Architekturen der leistungsstarken "diskreten" 16-bit-Systeme.

Alles in allem: Den anfänglichen trivialen Vergleich zwischen 8-bit- und 16-bit-Prozessoren sollte man lieber vergessen. "16 bit" bedeutet heute weit mehr als nur ein interner 16-bit-Bus. "16 bit" ist die Herausforderung der Computer-Bauelementehersteller überhaupt. Für einzelne Hersteller ist sie zur eigentlichen Strategie ihrer Produktionspolitik geworden. Hier wird mit einem extremen Einsatz an Material und menschlicher Arbeitskraft kalkuliert.

Die Produkte des 16-bit-Bereichs im Bezug auf Bauelemente läßt einen Vergleich mit 8-bit-Mikroprozessoren nicht mehr verhältnismäßig erscheinen. Die bekannten 8-bit-Systeme (typisch 8080) sind gegen die neue Generation der 16-bit-Einchip-CPUs wirklich nichts anderes als Spielzeug.

1.2 ENTWICKLUNGSGESCHICHTE UND STRUKTURWANDEL

Die **Entwicklungsgeschichte** der Datenverarbeitung ist in einen flachen und einen sehr steilen Abschnitt zu kennzeichnen. Der flache Abschnitt besagt einfach, daß es einer sehr langen Zeit bedurfte, bis jeweils ein nennenswerter Fortschritt erzielt wurde. Kennzeichnend ist weiter, daß der Übergang vom flachen in den steilen Entwicklungsabschnitt relativ abrupt erfolgte, – ganz im Gegensatz zu anderen physikalischen Entwicklungen. Man sollte überdies die Datenverarbeitung keinesfalls als eine rein logische Entwicklung aus der Mathematik her sehen. Die Datenverarbeitung ist nicht nur eine Angelegenheit des dualen Zahlensystems, des tetradischen Systems oder der numerischen Mathematik. Die Physik spielt hier gleichermaßen eine bedeutende Rolle. Der relativ abrupte Übergang in eine steile und sehr steile Entwicklungsphase ist von dem ersten, funktionierenden Elektronenrechner markiert worden. Dieses amerikanische Röhren-Monstrum namens ENIAC (Electronic, Numerical Integrator and Calculator) wurde an der Universität von Pennsyvania aufgebaut. 1945/46 ist dieser Rechner von J.M. Brainerd, J.P. Eckert, J.W. Mauchly und H.H. Goldstine konstruiert und fertiggestellt worden. Er galt zugleich als Prototyp einer Anlage der sogenannten ersten Generation, die ausschließlich mit Elektronenröhren und elektromechanischen Relais bestückt war. Er mußte über ein Steckfeld programmiert werden. 20 parallel angeordnete Akkumulatoren boten 20 zehnstelligen Dezimalzahlen Platz. Man erreichte mit dem ENIAC die Durchführung von etwa 5000 Additionen pro Sekunde als Rechengeschwindigkeit. Mit dieser Maschine wurden zahlreiche praktische Erfahrungen grundlegender Art erworben. Sie wurde im Laufe ihrer Dienstzeit bis 1955 an einzelnen Elementen verändert. Immerhin war das doch eine ziemlich lange Zeit, wenn man davon ausgeht, daß sie einen größeren Saal für sich beanspruchte. 18.000 Röhren benötigen nun mal Raum, – und vor allem elektrische Leistung für den Betrieb. Etwa



Chip der 8080 CPU (Firma Siemens)

mit E.-Röhren begann
die elektronische Rechentechnik

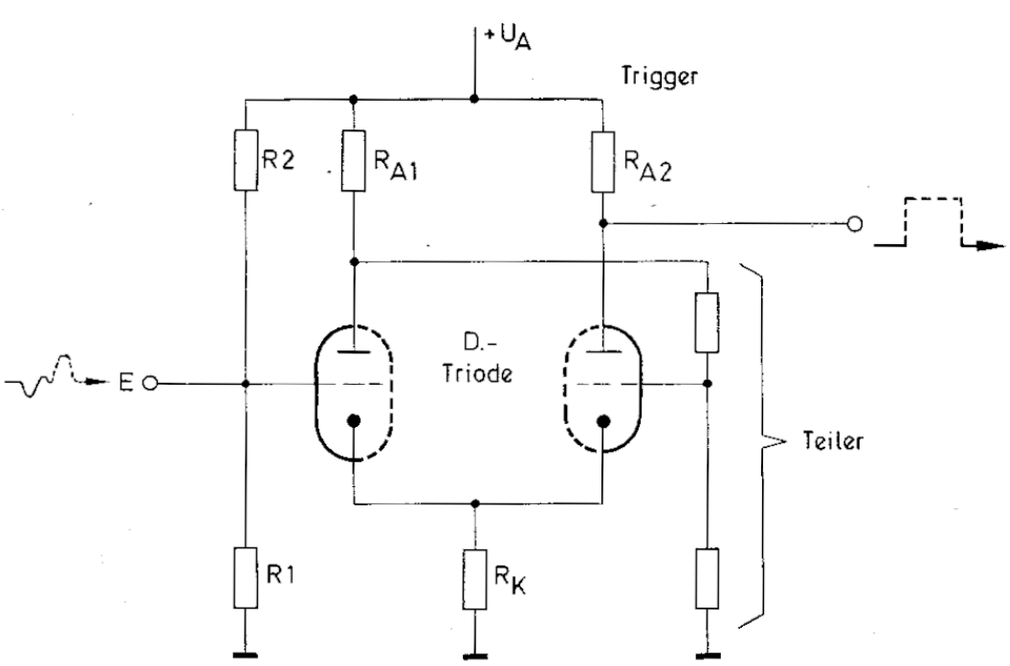


Bild 1.1.

180 kW waren dafür notwendig.

Der ENIAC brachte die Entwicklung der Datenverarbeitung entscheidend ins Rollen. Er ist wiederum ein Beweis dafür, daß technische Entwicklungen erst durch das Experiment leben. 1942 begannen Mauchly und Eckert mit der Ausarbeitung der Pläne zu dem ENIAC. Daß er überhaupt gebaut werden konnte, war einem nicht unerheblichen Anteil physikalischer Erkenntnis zu verdanken. **Bild 1.1.** zeigt eine Schmitt-Trigger-Schaltung mit einer Doppeltriode bzw. zwei Trioden, die eine der Grundschaltungen des Rechners waren.

Vor dem ENIAC wurde ein Meilenstein der Computertechnik von Charles Babbage (1792-1871) gesetzt, der mit seinem Konzept einer programmgesteuerten Rechenmaschine zu einem unmöglichen, ihm feindlich gesinnten Zeitpunkt Unglaubliches leistete. Seine elementare Maschinenkonzeption lebt in den Rechnern der Gegenwart weiter. Zur Entwicklung der heutigen Rechner haben noch eine Reihe wichtiger Begebenheiten und damit verbundene Persönlichkeiten beigetragen (Blaise Pascal, G.W. Leibniz, Konrad Zuse . . .). Betrachtet man das Jahr 1945 gewissermaßen als Schnittstelle zweier Entwicklungsabschnitte, so besteht zwischen der ENIAC und dem Thema dieses Buches, dem 16-Bit-Prozessor Z 8000 nur der geringe Zeitraum von 35 Jahren. Eine Menschengeneration trennt mehrere Generationen von Computern. Eine vergleichbare Extensität einer Entwicklung ist beispielsweise noch im Bereich der Molekular-Biologie bzw. der Genetik zu beobachten.

Die Entwicklung weiterer Rechner war in den 50er Jahren insbesondere von der Anwendung geprägt, dem Automaten vielstellige Zahlenrechnungen zu übertragen. Rechnen stand in den ersten Jahren im Vordergrund. Am Anfang der Entwicklung numerischer Algorithmen standen die vier Grundrechnungsarten: Addition, Subtraktion, Multiplikation und Division. Die damaligen Maschinen erlaubten diese Operationen keinesfalls in einem Schritt. Allein für die Addition mußte eine geeignete Vorschrift ausgearbeitet werden (Algorithmus), welche Schritte das Rechenwerk für diese Operationen auszuführen hatte. Trotzdem entwickelten sich hier relativ zügig geeignete Verfahren für die einzelnen Maschinen. Erste Anzeichen einer Softwareentwicklung waren festzustellen. Diese Softwarearbeiten wurden im Bereich der Rechnerarithmetik erstellt. Sicherlich, die Beschreibungen krankten teilweise an der Uneinheitlichkeit der Darstellung. Sie waren durchweg auf ein bestimmtes Maschinenkonzept bezogen. Eine zusammenfassende Übersicht dieser elementaren Algorithmen war zumindest in den 50er und Anfang der 60er Jahre praktisch nicht vorhanden. Auf dem im Jahre 1954 vorgestellten Großrechner des Typs 704 der Firma IBM wurden eine Reihe bedeutender Arbeiten erstellt. Dieser Rechner addierte damals schon zwei dezimale Summanden mit 12 Stellen im Mikrosekundenbereich. Es wurden über die Grundrechnungsarten hinaus die verschiedenen Iterationsverfahren angewandt, die zur näherungsweisen Berechnung von trigonometrischen Funktionen, Integralen und linearen Gleichungssystemen dienten.

Mit der Entwicklung der Rechner hatte auch zweifellos die numerische Mathematik ihren großen Aufstieg. Sie liefert die Regeln und Beweise der vielfältigen Reihenentwicklungen zur Lösung mathematischer Aufgaben, die für eine Lösung mit dem Rechner aufbereitet werden sollen. Summen und Produkte sind für den Rechner nun mal kein Problem. Iterationsverfahren sind nichts anderes als das mehrfach aufeinander folgende Anwenden verschiedener elementarer Algorithmen (der Grundrechnungsarten) zur Berechnung algebraischer und transzendenter Ausdrücke (oder Gleichungen). Für lineare Gleichungssysteme kann man z.B. den Gauß'schen Eliminationsalgorithmus, den Gauß-Jordan-Prozeß, die Dreieckszerlegung oder das Ortogonalisierungs-Verfahren über eine wählbare Transformationsmatrix anwen-

den; für die numerische Integration entsprechend die Newton-Cotes-Formel, die Euler-MacLaurin'sche Summenformel, der Extrapolations-Algorithmus oder die Romberg-Integration. Nebenbei sei hier nur bemerkt, daß man heute solche numerische Aufgaben leicht mit einem programmierbaren Taschenrechner wie den TI 59 von Texas Instruments lösen kann.

Um nochmals auf das Datenformat einzugehen: Das anfängliche Rechnen mit großen Zahlen beeinflußte das Datenformat der Maschinen entsprechend nach höheren Werten hin. Je breiter das Datenformat der Maschine ist,desto schneller sind große Zahlen berechenbar. Es ist eine Frage des Aufwandes der Maschine. Was das 16-bit-Format betrifft, eine eigene Geschichte hat es eigentlich nicht. Es entstand aus Kompromissen zwischen Aufwand und Rechengenauigkeit. Das 16-bit-Format war schon in den 50er Jahren anzutreffen. Jedoch arbeitete man auch mit Formaten einer Nicht-Zweierpotenz. Obwohl mit der PDP 8 im Jahre 1965 ein Minicomputer mit einem 12-bit-Format vorgestellt wurde, ist das 16-bit-Format gegen Ende der 60er Jahre überwiegend als besserer Kompromiß anerkannt worden. Das 16-bit-System PDP 11 ist hier sicher einer der Wegbereiter. Auch in dem Ende der 60er Jahre immer stärker aufkommenden Bereich der Prozeßrechner setzte sich das 16-bit-Format mehr und mehr durch. Das IBM-System /7 war ein für Anfang der 70er Jahre leistungsfähiges 16-bit-System mit 16 K Worte und einer 512 K Worte umfassenden Plattenspeichereinheit. Sie wurde u.a. für NC-Steuerungen von Werkzeugmaschinen benützt. Erstmals wurden hier Magnetplatten-Speicher statt des umständlichen Lochstreifens als Steuerdatenquelle eingesetzt.

Zusammenfassend kann hier festgestellt werden: Ende der 60er, Anfang der 70er Jahre hatten Rechner mit einem 16-bit-Datenformat eine gewisse Schlüsselstellung in der Computertechnik erreicht.

Ein zügiger **Strukturwandel** in der Datenverarbeitung war schon von je her vorhanden. In wenig anderen Branchen sind Produkte so kurzlebig wie in der Computerbranche. Das ist ihr typisches Charakteristikum. Der Begriff Strukturwandel sollte hier noch etwas präzisiert werden. Im Zusamenhang mit dieser Beschreibung ist er auf die letzten 6 bis 7 Jahre bezogen. Ab etwa 1973 machte sich diese Branche eine Aggressivität zu eigen, von der sie heute geprägt ist. Mit dem Aufkommen der Mikrocomputer wurde eine bereits erwähnte sehr steile Entwicklung eingeleitet. Die schnelle Weiterentwicklung der Integrationstechnik war der tiefere Grund dieses Strukturwandels. Wie wir aus der jüngsten Vergangenheit wissen, haben die integrierten Schaltungen die Rechnerhardware nachdrücklich beeinflußt. Mikrocomputer und die etablierten 16-bit-Systeme profitierten hier gleichermaßen. Den Computer-Bauelementeherstellern ist in diesen Jahren immer mehr die Position des Systemdesigners zugekommen. Der Entwurf der jeweiligen Systemfamilien (8080- und 6800-Familie; Z 80 CPU, PIO, SIO, CTC, DMA) in Form einzelner LSI-Bausteine ist der eingeschlagene und künftig streng weiterverfolgte Weg. Hierin sollte keinesfalls eine negative Entwicklung gesehen werden. An Hand des übersichtlichen Aufbaus eines Mikrocomputers mit diesen Bausteinen wurde das eigentliche Verständnis für Computer doch sehr zum Vorteil verändert. Die eindeutigen Funktionen einzelner Bausteine ermöglichen diesen klar gegliederten Aufbau. Die Eingrenzung einer bestimmten Intelligenzstufe in einem dieser Bausteine, voran der CPU, gibt den Rahmen zum Verständnis eines Elements. Es wird das Denken auf eine bestimmte Funktionsebene konzentriert.

In der folgenden Beschreibung wird jetzt ein entscheidender Denkschritt erläutert.

Die heutigen Mikrocomputerarchitekturen des mittleren Bereichs sind durchweg Monoprozessor-Systeme. Im Jahre 1975 wurde vereinzelt von verteilter Ver-

arbeitung, verteilter Intelligenz und mit fortschreitender Zeit von "distributed processing" gesprochen. Mit diesem Schlagwort wurde eine mehr nüchterne Umschreibung des ziemlich brisanten Begriffs "Rechnerintelligenz" geprägt. Trotzdem, die Begriffe treffen den Nagel schon auf den Kopf. Blickt man nochmals ein paar Jahre zurück und vergleicht die Rechnersysteme IBM 360 und IBM 370, so repräsentiert das 360er-System ein klassisches Monoprozessor-System. Die 370er-Maschine jedoch wurde schon in Richtung auf Multiprozessor-Architektur hin konzipiert. Greifen wir nochmal als Beispiel das System 6 von Honeywell Bull auf: Wenn diese Maschine entsprechend ausgebaut ist, wird sie zweifellos zu einem Multiprozessorsystem höherer Prägung. Hier werden die Ausmaße bereits sichtbar, welche Rechnerarchitekturen die 80er und 90er Jahre bringen werden. Das ist der Strukturwandel im tieferen Sinn, der hier gemeint ist.

Kommen wir nun endlich zur Gegenwart. Welche Zielsetzung haben diese großvolumigen VLSI-Bausteine, die 1979 vorgestellt wurden? Ihre Zielsetzung liegt doch einzig und allein darin, den begonnenen Weg eines extensiven "Mensch-Maschine-Dialogs" immer mehr zu verfeinern. Die Beschreitung dieses Weges wird nun mal künftig über Mehrfach-Rechnerarchitekturen geschehen. **Bezieht man jetzt genauer den Typus der Z 8000 CPU auf diese Eigenschaft hin, so wurde ihre Architektur mit einem Augenmerk auf künftige Parallelrechner-Systeme vorbereitet.** Man ging hier gründlich zu Werke. Spezielle Hard- und Softwarefacilities deuten hier klar darauf hin.

Die CPU und durchweg alle größeren Peripheriebausteine, die in diesem Buch vorgestellt werden, gehören zur absolut höchsten Kategorie, die die Computer-Bauelementehersteller gegenwärtig in der Lage sind herzustellen. Bausteine dieses Kalibers werden die Rechnerarchitekturen mittlerer und auch großer Systeme für die nächsten Jahre bestimmen. Der Zyklus bis zur Markteinführung eines 32-bit-Prozessors wird etwa bis 1987 dauern. Die Entwicklung in diese Richtung wird jedoch nicht ausschließlich technologische Barrieren zu überwinden haben. Die Hersteller bieten ihr Bestes. Man sollte die Herausforderung der Hersteller annehmen.

Systeme mit Bausteinen der Z 8000-Familie werden von Menschen entworfen werden, die einen Rechner der dritten Generation möglicherweise nicht zu Gesicht bekommen haben. Es ist nicht widerspruchslos einleuchtend, wenn von erfahrenen Computerfachleuten, auch "alte Hasen" genannt, verlangt wird, künftige Computer-Hardwaredesigner müßten den ersten Rechner verstanden haben, um an den Entwurf eines Parallelrechners herangehen zu können. Diesen Designern sind bereits eine Reihe von Überlegungen abgenommen worden. Es ist keinesfalls schwieriger, mit den in diesem Buch vorgestellten Bausteinen einen echten 16-bit-Hochleistungsrechner zu entwerfen, als mit antiquierten Bausteinen einen älteren Rechner hochzutrimmen. Das ist auch eine Konsequenz zum Aspekt Strukturwandel.

Der eingeschlagene Weg eines künftig noch besseren Mensch-Maschinen-Dialogs hat für die gegenwärtigen Systeme die Konsequenz zur Folge, die Verwaltung des Systems (sein Betriebssystem) noch weiter aufzublähen. Mehrfachbenutzer-Fähigkeit, Datenfernübertragung (Rechnerverbundnetz-Option) und der ausgiebige Dialog mit der am Bildschirmterminal sitzenden Person erfordert für die Maschine ein immer mehr zunehmendes Handling mit dem Betriebssystem. Diese Vorgänge sind etwa vergleichbar mit dem immer größer werdenden "Wasserkopf" der Verwaltung unseres Sozialstaates. Betriebssysteme sind gut und notwendig. Doch man übersieht hierbei, daß mit dieser Software das Anwenderprogramm nicht gelöst wird. Nicht selten benötigen große Rechner über 50 % der Maschinenzeit nur für "Verwaltungs"-Aufgaben, um den Betrieb der Maschine in Gang zu halten. Was ist dagegen zu tun? Allein

die Überlegung zum Parallelrechner oder Multiprozessor-System löst das Problem noch nicht. **Erst der gezielte Einsatz verteilter Intelligenz mit außerordentlich hoch entwickelter Hardware wird den Wunsch nach verbesserter Dialogfähigkeit mit der Maschine über ein maßvoll ausgebautes Betriebssystem lösen helfen.** Diese Hardware gibt es. Zentral- und Entlastungsprozessor kennzeichnen künftige Computerarchitekturen. Entlastungsprozessoren werden speziell zur Verwaltung der Peripherie eingesetzt werden; für bestimmte Schnittstellen je einen. Einen Interrupt zur Anmeldung eines Datentransfers in einen Pufferspeicher wird nicht die Host-CPU, sondern der Entlastungsprozessor bekommen. Darüberhinaus werden sie auch für allgemeinere Aufgaben wie z.B. mathematische herangezogen werden.

Dies ist alles unter dem Gesichtspunkt erhöhter Maschinenleistung zu sehen, die mit sehr gut zusammenwirkenden Hardware-Elementen und einem kleinen, lokalen Betriebsprogramm erreicht wird. Das eigentliche "main frame"-Betriebssystem wird diese kleinen Programme nicht mehr enthalten. Was ist das Ziel dabei? Ganz einfach: Der Zentralprozessor hat viel mehr Zeit für den Anwender. Er wird das Programm des Anwenders schneller bewältigen.

In einem Ingenieurbüro hat der Chef auch Spezialisten an der Hand, die ihm einige Aufgaben (zuverlässig) abnehmen. Bei den Programmiersprachen wird es bezüglich der Dialogfähigkeit auch noch einiges zu verbessern geben. Seit Beginn der Entwicklung der Programmiersprachen im Laufe der 50er Jahre wurden etwa 1000 Varianten entwickelt. Die Standardsprachen ALGOL 60/68, BASIC, COBOL und FORTRAN IV (ANSI-F. X3.9-1966), die vier wichtigsten, sind problemorientierte Sprachen. Klammert man Basic bedingt aus, sind für größere Aufgaben zum Teil schwierige Programm-Erstellungsverfahren notwendig. Die Problem-Umsetzung könnte einfacher sein. Im Detail ist das jedoch Erfahrungssache. In Basic läßt sich sowohl gut strukturiert wie interaktiv programmieren, d.h. in Form eines Dialogs (Frage und Antwort). Dies erfordert im allgemeinen längere Erstellungszeit der Programme und mehr Speicherplatz.

Es geht im Grunde darum, eine einfache Umsetzung des Problems zu erlangen. Die blockstrukturierte Programmiersprache PASCAL ist ein weiterer Fortschritt in Richtung einfacherer Umsetzung. Pascal erlaubt eine logisch folgende Strukturierung der Aufgabe mit weniger Speicherplatz als durchweg alle anderen problemorientierten Sprachen.

Rechner werden heute im kommerziellen technischen Bereich besonders intensiv zur Simulation technischer Anlagen (bis zu großen energieproduzierenden Anlagen) eingesetzt. In der Automation spielen Regelsysteme eine große Rolle. Für diese Belange wurde eine prozeßorientierte Sprache namens PEARL entwickelt (**P**rocess and **E**xperiment **A**utomation **R**ealtime **L**anguage). Sie besteht aus einzelnen Programm-Moduln, die vier Grundstrukturen eines Prozesses beschreiben. Diese Moduln sind in System- und Problemteil gegliedert und werden für einen spezifischen Anwendungsfall wahlweise hintereinander gekettet. Daraus ist leicht erkennbar, daß die Programm-Moduln im Speicherfeld verschiebbar sein müssen. Die variable Segmentadressierung und -verschiebung der Z 8000 CPU bietet hier wie geschaffene Voraussetzungen. Darauf wird im zweiten und fünften Kapitel noch genauer eingegangen.

Der Einsatz gegebener und noch neu zu entwickelnder Programmiersprachen ist letztlich von der Anwendung abhängig. Das Maß des Dialogs mit dem Computer ist ferner sehr von dem Typus der Person abhängig, die mit ihm arbeitet. Zusammenfassend kann über den Strukturwandel festgehalten werden: Die modernen VLSI-Bausteine werden in der Rechnertechnik für noch transparentere Architekturen sorgen und im Zusammenhang Zentral-Entlastungsprozessor eine Alternative zu Großrechnerkonzepten bringen.

1.3 EINCHIP 16-BIT-PROZESSOR ODER MEHRCHIP 16-BIT-PROZESSORSYSTEM: EINE STREITFRAGE ?

Man muß heute und sicherlich noch längerfristig zwischen zwei grundsätzlichen Kategorien einer Zentraleinheit (CPU) unterscheiden. Die Z 8000 CPU ist **ein** Computerbaustein mit **einem**, unabdingbar festen Befehlssatz. An ihm gibt es nichts zu korrigieren. Insbesondere diese CPU mit ihrer Random-Befehlsdekodierung ist hier streng definiert. Die Befehle des Z 8000 besitzen Makrostruktur, d.h. sie sind überwiegend aus einer größeren Zahl von elementaren Maschinenschritten entwickelt worden. Der typische Z 8000-Befehl CPSDR (Zeichenkettenvergleich, Adreßzähler erniedrigen um 1, – und das wiederholend) muß erst durch eine Verknüpfung einzelner Maschinenschritte gebildet werden. Bei Einchip-CPUs ist das immer so. Der Anwender merkt hier nichts davon.

Die zweite Kategorie von Zentraleinheiten besteht aus einer größeren Anzahl von Bausteinen. Eine Zentraleinheit besteht primär aus dem Steuerwerk (Befehlsdekodierung), dem Rechenwerk (ALU) und den Rechenwerksregistern. Hat das Rechenwerk eine bestimmte Mikrobefehlsstruktur (addieren, Komplement, UND, ODER . . .), so besteht die Funktion des Steuerwerks nur darin, eine bestimmte Folge dieser Mikrobefehle an die Rechenwerklogik heranzuführen. Die Erzeugung der Folge wird mit einem Sequenzer generiert. Es ist die Folge von Mikrobefehlen, die vergleichsweise erst einen typischen Maschinenbefehl ergeben, den man von den Mikroprozessoren her kennt. Der Sequenzerbaustein adressiert den Mikroprogrammspeicher nach entsprechender Folge. Aus dem Mikroprogrammspeicher werden die Mikrobefehlcodes für die ALU nacheinander angelegt. Nach jedem ausgeführten Mikrobefehl erhält der Mikroprogrammspeicher vom Sequenzer wieder eine neue Adresse zur Ausgabe eines neuen Mikrobefehlscodes.

So geht das. In **Bild 1.2** ist der Zusammenhang einer Zentraleinheit dieser Form dargestellt. Man sieht hier die Rückwirkung der Statusbits des Rechenwerks (ALU 2901, 2903) über das Statusregister (vergleichbar zu Flags in μPs), Steuer-PROM zum Sequenzer. Es handelt sich um Bausteine der Reihe Am 2900 der Firma Advanced Micro Devices, die in Slice-Architektur konzipiert sind. Das heißt nichts anderes als daß die Bausteine von ihrer Grundformatbreite von 4 bit auf jede Datenformatbreite erweitert (kaskadiert) werden können. Die 2900er-Bausteine zeigen überdies eine Zweiadreßstruktur, die bei Mikroprozessoren nicht üblich ist. Die beiden 4-bit-Adressen werden benötigt, um die Operanden an die ALU anzulegen (alles intern im 2901). Das aus den beiden Adressen angesteuerte Zwei-Port-RAM bildet eine Struktur mit der ALU, die bisher jedem Mikroprozessor überlegen ist. Daran ändert auch die hochgradige Akkumulator-Registerstruktur des Z 8000 nichts, weil das Anlegen der Operanden an die Z 8000-ALU sequenziell erfolgt. Die Spezialisten wissen das. Darüberhinaus bearbeitet die schnelle TTL Logik der 29er eigen entworfene Makrobefehle (Mikrobefehlsfolgen) von vorneherein schneller als der Z 8000. Es sind auch keine gleichen Voraussetzungen, von denen man hier ausgeht.

Aus der kurzen Erklärung über "Bit-Slices", so nennt man mikroprogrammierbare Rechnerbausteine, ist klar zu erkennen, daß der Aufwand für eine 16-bit-Maschine **beträchtlich** größer ist, wenn sie mit Bausteinen dieser Art entworfen werden soll. Außerdem muß anschließend ein Makrobefehlssatz in Form einer Sequenz erst entworfen werden. Daß das nicht einfach ist, kann man sich vorstellen. Jedoch, superschnelle 16-bit-Computer sind in dieser Art aufgebaut. Hier kann ein Z 8000-System teilweise nicht mithalten, was die Schnelligkeit der Programmausführung angeht.

Die Firma AMD bietet mit dem Am 2903

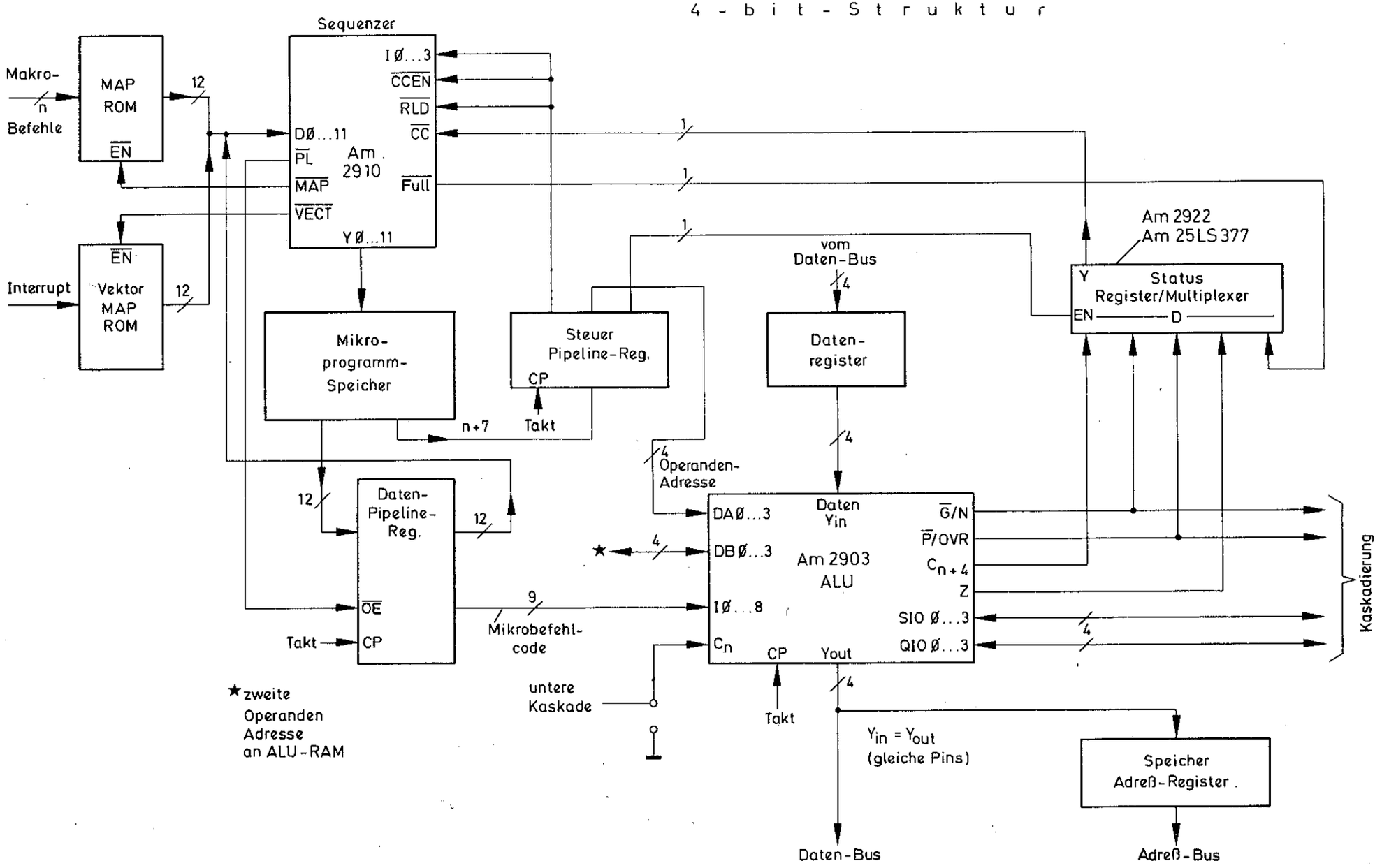

Sequenzer
Makro-
n
Befehle
MAP
ROM
EN
12
IØ...3
CCEN
RLD
DØ...11
PL
Am
2910
CC
MAP
Full
VECT
YØ...11
Interrupt
EN
Vektor
MAP
ROM
12
Mikro-
programm-
Speicher
Steuer
Pipeline-Reg.
CP
Takt
n+7
vom
Daten-Bus
4
Daten-
register
Am 2922
Am 25LS377
Y
Status
Register/Multiplexer
EN
D
4
Operanden-
Adresse
12
Daten-
Pipeline-
Reg.
12
DAØ...3
Daten
Yin
G/N
P/OVR
DBØ...3
Am 2903
ALU
Cn+4
Z
IØ...8
SIO Ø...3
QIO Ø...3
OE
9
Mikrobefehl-
code
Takt
CP
Cn
CP
Yout
untere
Kaskade
Takt
Yin = Yout
(gleiche Pins)
Kaskadierung
★zweite
Operanden
Adresse
an ALU-RAM
Daten-Bus
Speicher
Adreß-Register
Adreß-Bus

Bild 1.2.

einen ausgesprochenen Hochleistungs-Slice an, der eine Zweierkomplement-Multiplikation (ohne Vorzeichen) und -Division ermöglicht. Dieser Baustein ist in wenigen 16-bit-Superrechnern mehrfach enhalten. Man sollte jedoch hier nicht vergessen, daß Rechner dieser Geschwindigkeitsklasse auch im Preis eine Obergrenze bilden und in spezifischen Bereichen wie extrem schneller Echtzeit-Simulation ihr Anwendungsfeld finden.

Die zweifellos "gewaltigsten" Rechnerbausteine bezüglich Geschwindigkeit sind die 8-bit-Slice-Produkte des Typs F 100 K der Firma Fairchild. Diese ultraschnellen ECL-Bausteine (Emitter gekoppelte Logik) wurden gegen Ende 1978 vorgestellt und haben eine beachtlich hohe Dichte von etwa 1000 Gattern und mehr auf einem Chip. Der Isoplanar II-Prozeß ermöglichte eine **Gatterlaufzeit von 0,75 Nanosekunden!!!**

Dieses ECL-Bausteinsystem besteht aus vier Kernelementen:

- Die Adreß-/Daten-Interface-Einheit ADIU, die sowohl als zentrale ALU oder als Memory-Management-Einheit benutzt werden kann. Sie beherrscht BCD- und Binär-Arithmetik.
- Der Multifunktions-Netzwerkbaustein MFN kann als Zentralregister-Stack adressieren, den Bus bidirektional multiplexen, als Register-/Speicher-Adreßwerk oder Speicher-Multiport arbeiten.
- Das Parallel-Stack DAS hat zwei getrennte Daten-E/A-Kanäle mit je 9 bit Breite und einen Speicher mit 32 x 9 bit Worten inklusive Schreib-/Lese-Steuerung.
- Der programmierbare Schnittstellen-Baustein PIU (vollkommen asynchroner Betrieb) mit 48 mA offenem Kollektorausgang ist für bidirektionale (unidirektionale) E/A-Ports oder bidirektionalen Zugriffsbetrieb ausgelegt.

(F100200ADIU,F100221MFN,F100222DAS und F100223PIU)

Werden acht ADIUs kaskadiert und mit der Carry-Look-ahead-Einheit F 100179 verknüpft, wird eine 64-bit-Addition in 25 ns ausgeführt. Das ist jenseits üblicher Rechner-Netzwerke. Aus diesem "Holz" sind bzw. werden künftig Großrechner "geschnitzt". Der Groß-EDV-Hersteller NCR wendet die F 100K-Bausteine in den Großsystemen V-8600 an. Diese Bausteine des absoluten oberen Geschwindigkeitslimits sind in ein System vielfach redundant einzusetzen. Beim Design dieser Bausteine haben nicht nur der Hersteller selbst, sondern auch Anwender, wie in diesem Fall die Firma NCR, mitgewirkt. Die F 100K-Bausteine sind sehr teuer. Eine ADIU kostet etwa 700 DM. Bedenkt man, daß neben einer Vielzahl dieser Bausteine die schnellsten Schreib-/Lesespeicher verwendet werden müssen (Zugriffszeit etwa 10 ns), ist hier leicht einzusehen, in welchen Preisgefilden sich diese Rechner bewegen; trotz intensiver Normierungsbestrebungen. Hier kann dagegen auch ein Z 8000-System spielerisch wirken.

Um auf die Streifrage zu kommen: Eine Streifrage im üblichen Sinn gibt es hierzu nicht. Es wird zwischen Einchip-16-bit und Slice-Mehrchip-16-bit mit zweierlei Maß gemessen. Der Aufwand für Slice-Rechner ist generell so hoch, daß hier nur eine spezifische Anwendung oder ein spezielles Marktsegment diesen extremen Aufwand rechtfertigt. Betrachtet man nüchtern genug die Verhältnismäßigkeit des Einsatzes und die wesentlich kostengünstigere Herstellung des Einchip-Systems, kann das doch zum Umdenken veranlassen. Zumal ist über die Software bei Einchip-Systemen der relative Vorteil der Geschwindigkeit von Slice-Rechnern korrigierbar. Schon die alleinige Verfügbarkeit des Z 8000 wird so manchen Slice-Rechnerhersteller im 16-bit-Bereich zum Umdenken veranlassen. Bei Großrechner-Produzenten wie NCR und Control Data, deren Erfahrung in der ECL-Technologie begründet liegt, werden Veränderungen im 16-bit-Bereich sicherlich keine Umdenkprozesse einleiten.

Die Einchip-16-bit Zentraleinheit Z8000

2

2.1 EIN WENIG ENTWICKLUNGSGESCHICHTE

Dieses Buch entstand aus einer echten Leidenschaft zu einem phantastischen Computerbaustein. Der erste Abschnitt sollte, auf einigen Seiten zusammengefaßt, den Weg aufzeigen, wie es zu dieser Beschreibung der Z 8000-CPU kam. Es sollte auch hier zum Ausdruck kommen, daß der Autor kein Computer ist. Die weiteren Abschnitte dieses Buches lassen keinen Raum für irgendwelche menschlichen Züge zu. Hier wird "grimmige" Computertechnik geboten. Deshalb verzeihen mir bitte die verehrten Leser, wenn in diesem Abschnitt (und nur in diesem) in der "Erzähl"-Form geschrieben wird. Es läßt sich hier leider nicht vermeiden. Für den Leser, der menschliche Züge in einem Fachbuch für falsch am Platz hält, wird ab dem Abschnitt 2.2 die richtige (und auch angebrachte) technische "Tonart" angeschlagen, die für einen Computerbaustein der obersten Kategorie angebracht ist.

Es begann eigentlich ziemlich nüchtern. Die Fachausstellung "electronica 78" in München im Herbst des Jahres war Ort und Zeitpunkt des ersten Zusammentreffens zwischen des Z 8000-Prozessors und mir. Um korrekt zu sein, nicht mit der CPU selbst, sondern nur mit einer englischen Informationsschrift namens "AmZ 8000 Microprocessor Family" am Stand der Firma Advanced Micro Devices. Die CPU gab es zu diesem Zeitpunkt noch nicht. Ein kurzes Überfliegen der Broschüre riß mich keineswegs vom Stuhl, zumal auf der Seite vier gleich gegen die liebe Konkurrenz (PDP 11 und 8086) losgehämmert wurde. Ich muß hierzu auch bemerken, worum es bei dieser CPU geht. Die zweite, 71 Seiten starke Broschüre über Specifications schleppte ich einfach mit, ohne sie anzusehen. Der erste Kontakt war also keineswegs durchdringend.

Ich muß zugeben, ich habe mich zur damaligen Zeit ziemlich intensiv mit dem (und nur dem) anderen 16-bit-Prozessor beschäftigt, dem 8086 von der Firma Intel. An dieser CPU habe ich die beträchtliche Differenz zu 8-bit-Prozessoren kennengelernt. 29.000 Transistoren auf einem Chip waren für mich damals einfach etwas gewaltiges. Noch im Dezember des Jahres 1978 baute ich den Intel-Kit SDK 86 auf, freute mich über das Zustandebringen und eignete mir zu Weihnachten 78 die ersten praktischen Erfahrungen an. So ganz nebenbei nahm ich einmal die Z 8000-Specifications-Broschüre zur Hand und blätterte so ein wenig querfeldein. Es zündete immer noch nicht, denn ich hatte ihn immer noch nicht gut genug begriffen. Der 8086 saß noch zu tief, ganz einfach, weil noch eine gewisse Hemmschwelle gegenüber dem Z 8000 da war. Ich las nur hin und wieder kleine Notizen über das Superding Z 8000, welch futuristische Architektur usw., usw. . . Noch immer zündete es nicht. Aber jetzt! Es war so gegen Ende März 79, als endlich der Funke übersprang. Es war ein in unförmigem Englisch geschriebener Bericht über einen Vergleich der Z 8000-CPU mit dem 8086. Ich nahm mir die nötige Zeit, mit einer entsprechenden Vorbelastung auf den 8086, die Unterschiede kennenzulernen. Ich begriff jetzt endlich, was ich hätte schon längst begreifen sollen. Die Specifications-Broschüre wurde jetzt öfters in die Hand genommen. Ich begann nach weiterer Do-

kumentation über den Z 8000 zu suchen. Im ersten Halbjahr 79 gab es noch so gut wie nichts. Ein wesentlicher Anstoß wurde mir auf der IMMM in Genf gegeben (Juli 1979). Die IMMM ist eine noch junge Messe über Mikrocomputer und Komponenten. Ich bekam ein Exemplar eines Z 8000-Befehlssatz-Manuals am Stand eines Schweizer Zilog-Distributors. Es hatte noch das allseits bekannte, in der Elektronik-Industrie vielfach verwendete Entschuldigungs-Prädikat "Preliminary" fein säuberlich in Blau aufgedruckt. Das bedeutet so viel wie: Es darf noch verdunkelt werden. Von Verdunkeln konnte keine Rede sein. Hierin wurde schon ernsthaft "Befehls"-Farbe bekannt.

Im August 79 wurde bei der Firma Kontron, nahe bei München, ein erstes Seminar abgehalten. Ich bekam jetzt das PLZ/ASM Assembly Language Programming Manual ohne das diskrete "Preliminary" und eine Reihe weiterer kleiner Dokumentationen, z.B. über die Z 8010-MMU. Auf diesem Seminar konnte ich IHN das erste mal sehen. Auf dem Entwicklungs-Modul-Board in Zusammenhang mit dem auf einer Z 80-CPU basierenden Steuersystem wurde der kleine Bruder, die nichtsegmentierte Z 8002-CPU, für mich erstmals sichtbar, eingesetzt.

Die Computer-Fachausstellung "Systems 79" im September in München war ein weiteres Datum. Auf dem Stand der Firma AMD war für mich erstmals die AMD-Variante zu sehen. Das kleine 8002-System auf einer Platine ist an eine Tastatur angeschlossen, womit es komplett zu bedienen ist. Die ASCII-Eingabetastatur-Platine enthält eine alphanumerische LED-Zeile mit 20 Zeichen. Von diesem System war ich schnell begeistert. AMD bot zu dieser Messe das Processor-Instruction-Set-Buch und das Büchlein AmZ 8001/8002 Processor-Interface an.

Im Laufe der folgenden Wochen hat sich ein anfänglicher Gedanke zu einem realen Vorhaben herauskristallisiert. Der Z 8000 war mein Thema. Jede weitere theoretische Beschreibung festigte die Meinung zu meinem Vorhaben.

Die Entscheidung war so gut wie gefallen. Von diesem Zeitpunkt an war mir klar, was ich künftig zu tun hatte. In den folgenden Wochen opferte ich entsprechend private Zeit, den Z 8000 im Detail zu verstehen. Ich war auch künftig bereit, jegliche Sicherheit meiner Anstellung zu opfern, um genügend Zeit für diese Aufgabe zu haben.

Der Leser wird sich jetzt vielleicht fragen, warum diese CPU gewählt wurde.

Mein Interesse bei Mikrocomputern galt von Beginn an der Maschinenebene. Damit ist sowohl die Hardware des Systems (CPU-Speicher-Peripherie) als auch die Programmierung auf Maschinenebene (Assembler) gemeint. Das Arbeiten mit 8-bit-Microcomputern brachte die generelle Erfahrung mit sich, daß eine Softwareaufgabe auf Maschinenebene aus zwei Befehlsgruppen besteht. Die Befehle sind in beiden Gruppen aus dem allgemeinen, für die CPU typischen Befehlssatz gebildet und keineswegs streng getrennt, bezüglich ihrer Zeilennummer. Sie bilden eine logische, organische Einheit im Programm. Ein Programm besteht bekanntlich aus einer Reihe von Befehlen an den Prozessor, die sinnvoll aneinander gereiht, die Lösung einer Aufgabe ergeben. Sieht man die einzelnen Befehle als gewichtet an, bilden ihre gesamte Anzahl diese beiden Gruppen:

A Befehle mit dem Ergebnis einer realen Hinführung zur Lösung der Aufgabe

B Zubringer- oder Vorbereitungsbefehle, die die Ausführung eines Befehls der Gruppe A bewirken können.

Man muß die beteiligten Operanden erst holen, den Index setzen, umladen, neuladen . . . um z.B. den Vergleich, die Verschiebung oder die Arithmetik durchführen zu können. Vorbereitungsbefehle zu einem Befehl des Typs A sind untrennbar fest in die Programmbefehl-

Nummernfolge eingebaut. Die Befehle des Typs B sind problem- und maschinenabhängig. Ist ein A-Befehl ausgeführt, folgen wieder einige B-Befehle, um die nächste A-Operation vorzubereiten. A-Befehle sind nur problemabhängig. Man kann die A-Befehle sehr gut damit vergleichen, wenn man sich ein allgemeingültiges Flußdiagramm ansieht. In den einzelnen Blöcken sind die Operationen bezeichnet, die unbedingt zur Lösung hinführen. Vorbereitungsoperationen zu den Blöcken läßt man im allgemeinen weg, weil sie etwas verwirren oder die Maschinenstruktur zu sehr verdeutlichen.

Und das ist der springende Punkt. Die Struktur der Maschine (CPU intern) erfordert nun mal das Einhalten der Regel, sich auf ihre Register, ihre Befehle, ihre Flags, kurzum auf alles interne einzustellen. Bei der Programmierung in höheren Sprachen ist das selbstverständlich nicht notwendig. Auf Maschinenebene ist das so. Man muß damit leben. Bei der Lösung einfacher Aufgaben (monotone Strings, Blockverschiebungen, einfache 8-bit-Arithmetik wie ADD, SUB) ist das kein Problem. Schwierig wird es bei Gleitkommaoperationen oder der Berechnung von mathematischen Reihenentwicklungen. Wenn man nur ein Akkumulator-Register zur Verfügung hat, muß man ständig einen Teiloperand wieder auslagern, holen usw. Hier häufen sich unweigerlich Zubringeroperationen des Typs B, die eigentlich mit der Lösung der Aufgabe nichts zu tun haben. Man muß sie durchführen, weil die Bearbeitung des Programms der Maschinenstruktur untergeordnet werden muß. Deshalb ist vielleicht hier zu verstehen, wenn ich damit andeuten will, daß Programme solcher Art mit der einen oder anderen 8-bit-CPU sehr schwierig zu lösen sind und auch viel Maschinenzeit in Anspruch nehmen.

Die Z 8000 CPU eröffnet hier sehr viel freundlichere Perspektiven. Der breite Registersatz für universelle Anwendungen (Akku, Index, Stack) und die variablen, transparenten Maschinenbefehle helfen, die Operationen des Typs B teilweise stark zu verringern. Z 8000-Maschinenprogramme sind verständlicher aufzubauen. Auf der Basis der sehr anwenderfreundlichen Maschinenstruktur lassen sich viel schwierigere Aufgaben angehen als mit 8-bit-CPUs. Man kann Programme sowohl schnell als auch übersichtlich erstellen.

Absolute Grenze stellen die im Befehlssatz eingebauten vier Grundrechenarten dar. Es können Operanden bis 32 bit Länge ein Ergebnis bis 64 bit erzeugen, ohne daß abgeschnitten wird. Arithmetik mit jeweils einem Befehl, das spricht für sich.

Man muß es schon mal klar ausdrücken. Auch das Erlernen sehr einfacher Maschinenstrukturen erfordert Zeit. Daran kommt man nicht vorbei. Man sollte lieber die Zeit dafür nützen, eine Maschinenstruktur verstehen zu lernen, mit der man ein weites Spektrum von Anwendungen erfassen kann, bei denen zudem hohe zeitliche Anforderungen gestellt werden. Versteht man eine Maschine des Kalibers eines Z 8000, so ist das zweifellos ein begeisterndes Gefühl, weil man weiß, daß man sich an sehr schwierige Programmieraufgaben heranwagen kann.

Der Z 8000 und seine Hersteller

Der Z 8000-Chip wurde von der Firma Zilog entwickelt. Zilog hat 1976 mit der Z 80-CPU seiner Zeit Zeichen gesetzt. Der Z 8000 knüpft nicht an seinen Vorgänger an, wenn auch die mnemonischen Bezeichnungen der Z 8000-Befehle mit denen des Z 80 Ähnlichkeit aufweisen. Die Zielsetzung von Zilog war, zwischen 8 bit und 16 bit einen eindeutigen Trennstrich zu ziehen. Die 16-bit-CPU offenbart die extensive Denkweise bei Zilog. Der Z 8000 ist kein Ableger "alter Zöpfe", sondern ein eigenwilliges Produkt, dessen Eigenschaften sich an denen etablierter "Minicomputer" nicht nur orientiert. Der Datendurchsatz des Z 8000 A (6 MHz takt) übertrifft die vieler Minicomputer beträchtlich.

Die Firma AMD hat in einer frühzeitig eindeutigen Entscheidung den Zuspruch

für diese CPU erklärt. Diese kompromißlose Einstellung ist ein weiterer Beweis des hohen Standards dieser CPU. AMD genießt einen sehr guten Ruf unter den Halbleiterherstellern, der in einem extrem hohen Niveau der Qualität begründet liegt. Ein kleines Beispiel soll das verdeutlichen: In einem neutralen Test über Bi-FET-Operationsverstärker (Typenreihe LF 155 . . . 157) wurde von der Firma Plessey Microsystems eine Untersuchung über Zuverlässigkeit durchgeführt. Die AMD-Typen der LF-Reihe bewiesen noch bei über 200 °C Umgebungstemperatur ihre volle Funktionsfähigkeit über die getestete Zeit. Die hohen Qualitätsmaßstäbe bei der Fertigung ergeben für die Mehrzahl der Bausteine die Zuverlässigkeit nach der Spezifikation MIL-STD-883. AMD-Bausteine gehören bekannterweise zu den höchstqualitativen unter den analogen und insbesondere den Digitalbausteinen. Die Mathe-Prozessoren 9511 und 9512 sind ausgesprochene Exoten, für die es kein unmittelbares Äquivalent gibt. AMD ist kein Zweithersteller im üblichen Sinn. AMD hat das Design einiger wesentlicher Peripherie-Bausteine des Z 8000-Sytems übernommen, die mit Zilog ausgetauscht werden. Es bleibt hier nur zu wünschen übrig, daß die Zusammenarbeit beider Firmen Bestand hat und nicht nur eine kurze Liaison ist.

Als dritter Hersteller des Z 8000 wäre die in Italien beheimatete Firma SGS-Ates zu nennen, die auch als Hersteller des Z 80 und Peripherie bekannt ist. Japanische Halbleiterhersteller haben bereits eine feste Option auf die Z 8000-CPU angemeldet.

2.2. TECHNOLOGIE UND ARCHITEKTUR EINER RANDOM ORIENTIERTEN CPU

Die Z 8000-CPU sollte nach ihrer Konzeption eine Reihe von Eigenschaften in sich vereinigen, die von Hochleistungs-Minicomputern her bekannt waren. Im einzelnen wurden diese Eigenschaften nochmals verfeinert und präzisiert. Die bei der Entwicklung der CPU verfolgten Ziele sind, gewissermaßen als Bedingung, in folgenden Punkten zusammengefaßt:

- variable Segmentadressierung
- wählbare Adressierungsarten der bekannten Typen für einen großen Teil der Befehle
- vielseitiger Befehlssatz für Anwendungen in der on-line-Prozeßtechnik, der Textverarbeitung und des leichten Entwurfes von Compilern strukturiert orientierter Programmiersprachen (Dialog-Sprachen)
- mathematisch und logisch orientierter Befehlssatz
- vielseitiger Registersatz mit gleichartigen Universalregistern (Akku, Index, Stack) mit Möglichkeiten von 32-bit-Wortoperationen
- fein nivellierte Interruptstruktur
- hardwaremäßige Priviligierung spezieller Befehle
- eine einfach anzuwendende CPU-Steuersignalbehandlung (Status-Definition) für exaktes Timing des CPU-Busses
- eine einfach anzuwendende Behandlung einer CPU-Bus-Anforderung für DMA-Operationen
- ein möglichst kleines CPU-Gehäuse
- hardware- und softwaremäßige Vorkehrungen für den Einsatz der CPU in Multiprozessorsystemen
- eine CPU-Technologie, die eine wirtschaftliche Ausbeute von Chips aus dem Wafer garantiert

Diese sich teilweise widersprechenden Eigenschaften wurden einer willkürlichen 16-bit-Architektur in einem Opti-

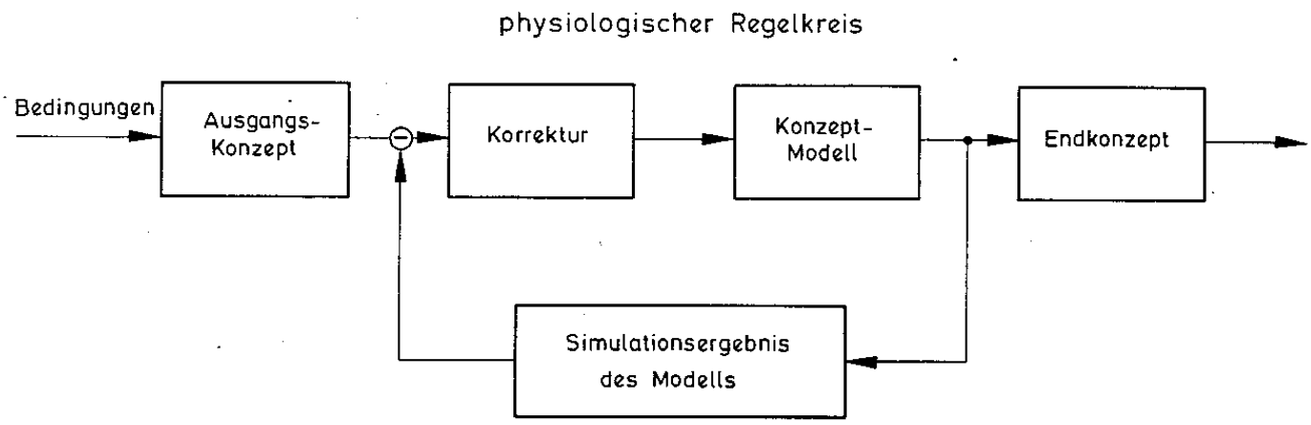


Bild 2.1

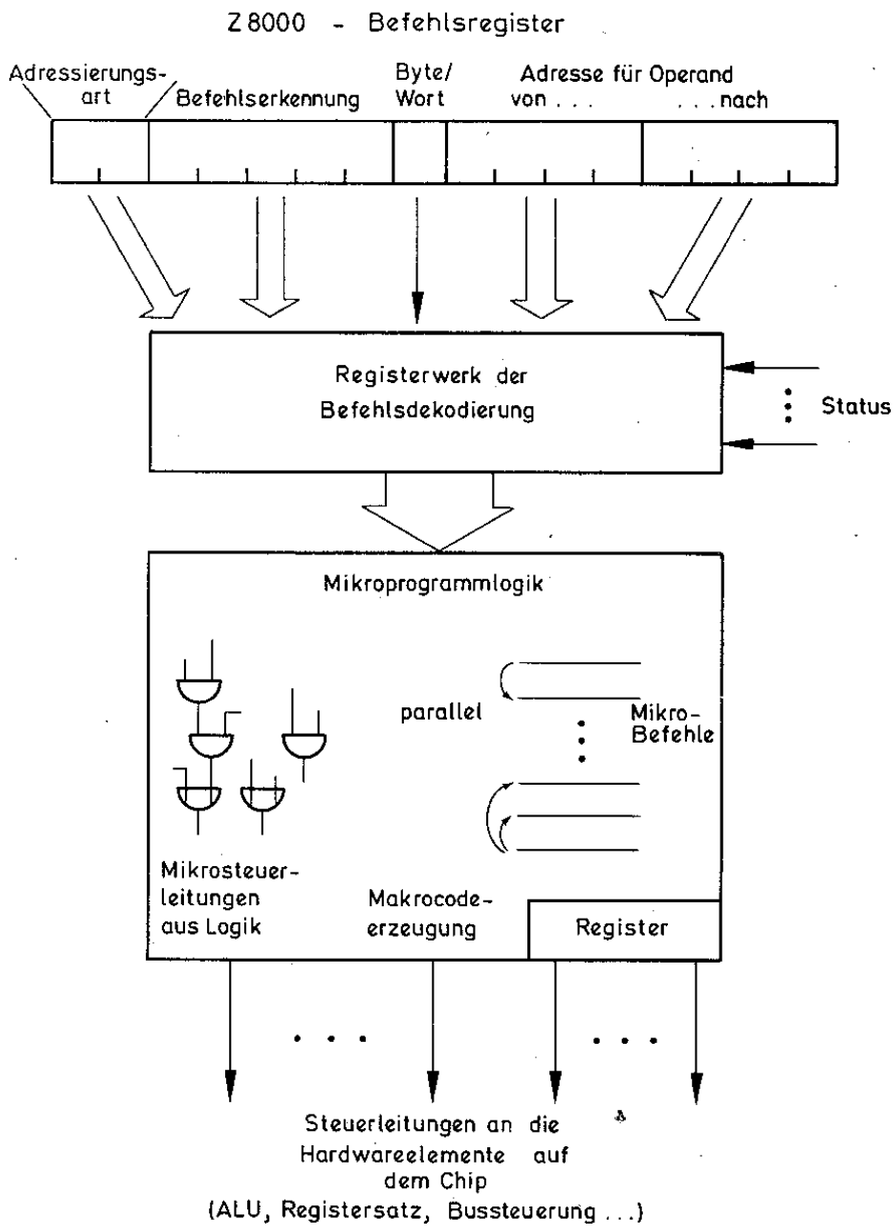


Bild 2.2

mierungsprozeß Schritt für Schritt angepaßt. Ganz so willkürlich war das Anfangskonzept nicht. Nach Fertigstellung des ersten Layouts hatte das Konzept der CPU schon sehr klar umrissene Details aufzuweisen. Dieses Ausgangskonzept war Gegenstand des Optimierungsprozesses. Man muß ein Konzept haben, das sich stetig verbessern läßt. Es war eine Art physiologischer Regelkreis **(Bild 2.1)**, über den sich der Annäherungsprozeß zu der angestrebten CPU-Architektur vollzog.

Eine Bedingung wurde frühzeitig festgelegt. Man wollte unter allen Umständen vermeiden, die CPU in ein 64poliges Gehäuse zu designen, weil diese großen Gehäuse sehr teuer in der Produktion sind. Um die obigen Bedingungen alle unter einen Hut zu bekommen und gleichermaßen die Chipfläche nicht zu sehr anwachsen zu lassen, mußte man den Weg einer Optimierung des Chip-Designs gehen. Optimierung bedeutet hier, so wenig wie nur irgend möglich Transistoren einsetzen. Man kann sich vorstellen, was hier von den Entwicklern des Z 8000 verlangt wurde. Was man sich nicht mehr vorstellen kann, ist, wie diese Aufgabe gut lösbar ist. Bedenkt man, daß nur ein paar Menschen aktiv an dem Design des Z 8000 beteiligt waren, ist es kaum glaubhaft, daß die Aufgabe überhaupt gelöst wurde.

An der Entwicklung des Z 8000 war der Japaner Masatoshi Shima maßgebend beteiligt. Dieses Genie unter den Mikroprozessor-Designern hatte das Kapital einer fundamentalen Erfahrung im μP-Design in sich. Er war an der Entwicklung der 4-bit-CPU des Typs 4004 und an den 8-bit-CPUs 8080 und Z 80 schon entscheidend beteiligt. Jede CPU war eine beträchtliche Steigerung seiner Art. Man konnte erwarten, daß er sich für seine erste 16-bit-CPU etwas besonderes einfallen läßt. Und er ließ sich auch etwas einfallen. Er wußte, daß am meisten Transistoren und Fläche für die Befehlsdekodierung und Mikrocode-Erzeugung auf dem Chip verbraucht wurden. Und genau da setzte er den großen "Sparhebel" an. Er entwarf eine optimierte Random-Logik für seinen Befehlsdekodierer. Nur die Eigenwilligkeit eines Shima bringt so etwas fertig. In **Bild 2.2** ist dargestellt, was damit gemeint ist.

Im Befehlsregister steht ein Wort (Befehl oder Teil davon). Die Dekodierung wird von einem festverdrahteten Logikkomplex erledigt, aus dem die diskreten Steuerleitungen zu den einzelnen Funktionsgruppen der CPU verbunden werden. Ein Mikroprozessor ist eine exakt terminierte Logik. Es muß bei einer Befehlsausführung gewährleistet sein, daß ein bestimmter Pfad freigegeben wird, wenn er gebraucht wird. Dies erfordert die Aktivierung einer bestimmten Zahl von Steuerleitungen (aktiv Low), die über Tri-State-Ausgänge den Pfad (und nur diesen) freigeben. Jede Leitung muß sitzen, schwindeln gibt es hier nicht.

Der Eingang ist das Befehlswort. Über die festverdrahtete Logik wird letztlich die Steuerung aller Funktionselemente bewirkt. Diese festverdrahtete Logik (auch Random-Logik genannt) repräsentiert das zentrale Steuerwerk der Z 8000-CPU. Sie ist deshalb so schwierig zu erstellen, weil einerseits durch die vielen Befehle die Bedingungen an sie eine beträchtliche Quantität erlangen und andererseits bit für bit die Bedingungen in einer Steuerleitung enthalten sein müssen. Gatter für Gatter muß hier verknüpft werden.

Es geht auch viel einfacher. Man setzt einen Mikrocomputer für diese Aufgabe ein. Dieser Mikrocomputer auf dem Chip des Mikroprozessors hat die Aufgabe, aus einem vorliegenden Befehlscode die feste Zahl von Steuerleitungen zu aktivieren, die für die Freigabe eines Pfades erforderlich ist. Das sind teilweise lumpige Vergleichsoperationen. Diese Methode nennt man Mikroprogrammierung. Sie wird in den meisten Mikroprozessoren als Befehlsdekodierwerk eingesetzt. **Bild 2.3** zeigt diese Art. Eine feste Tabelle von Befehlscodes steht einer ebenso festen Tabelle zu aktivierender

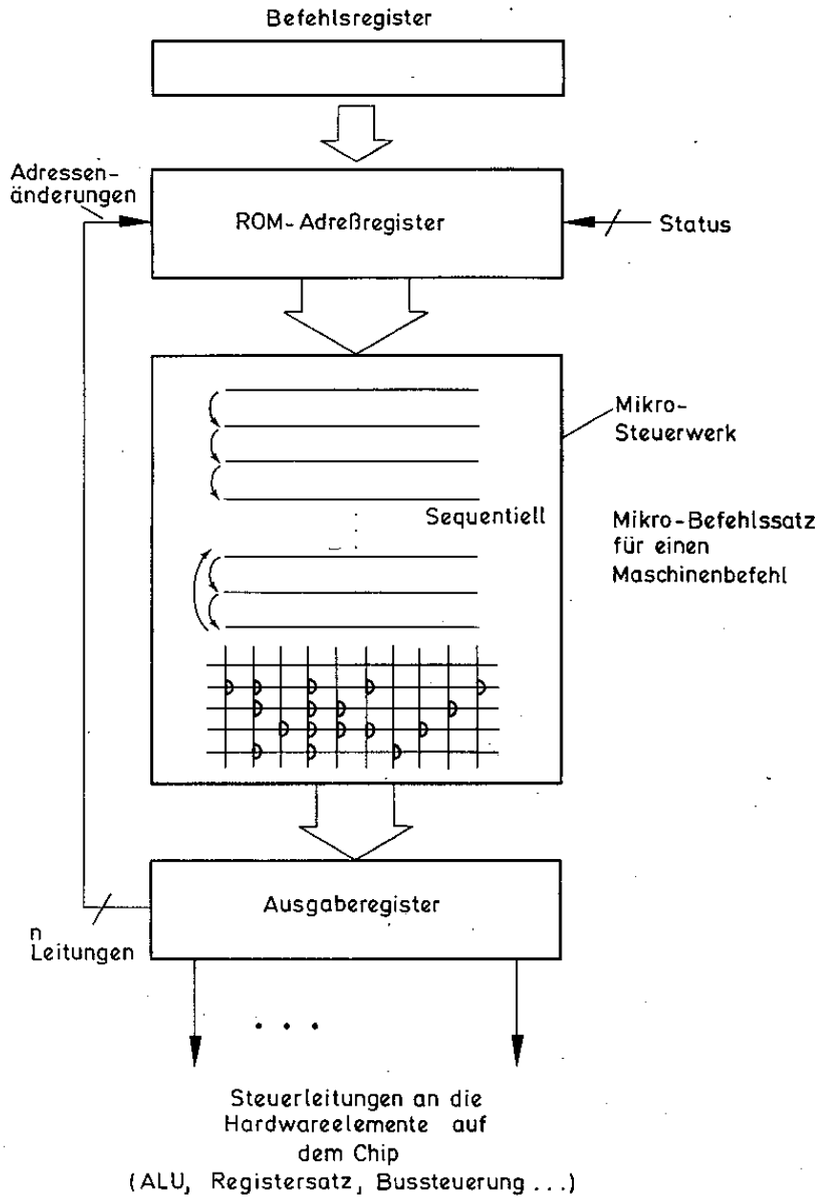


Bild 2.3

Steuerleitungen gegenüber. Sie hat einen Vorteil beim Design. Sie kann sehr gut mit einem Simulationscomputer wie die Sentry 7/8 auf Tauglichkeit überprüft werden.

Man muß die Mikroprogrammierung, genauer die Mikroprogramm-Steuerwerke, noch unter einem anderen Licht sehen. Sie wird deshalb so häufig bei Computern verwendet, weil sie relativ einfach zu erstellen ist. Darüber kann man wohl auch noch streiten, es sollte trotzdem hier so stehen bleiben. Um ihren Erfolg zu verstehen, muß man nochmals in der Vergangenheit wühlen. Worin liegt der Erfolg der IBM-Rechner unter anderem begründet? Es liegt zum Teil daran, daß sehr verschiedenartige Rechner immer den gleichen Maschinenbefehlscode verwenden. Viele Programme können sowohl auf kleinen als großen Maschinen laufen. Ein hohes Maß an Software-Kompatibilität bedingt zweifellos einen Erfolg. Erreicht wurde diese Anpassung durch die Mikroprogramm-Steuerwerke. Der Unterschied zwischen kleinen und großen Rechnern besteht in der Programmierung und im Mikrobefehlsformat dieser Mikro-Steuerwerke. Das Adreßregister adressiert die Mikrobefehle, die im Mikroprogrammspeicher stehen. Jeder Mikrobefehl definiert eine Anzahl möglicher Folgezustände und zugleich eine Anzahl spezifischer Steuerleitungen zu den CPU-Funktionsgruppen (man kann die aktivierten Steuerleitungen

auch mit Ausgangsvektoren bezeichnen). Die Folgezustände adressieren eventuell noch weitere Mikrobefehle. Die Summe der insgesamt daraus entstehenden Ausgangsvektoren (aktivierte Steuerleitungen) ergibt die Ausführung eines Maschinenbefehls. Jetzt sieht man auch ein, daß die Ausgangsvektoren sequentiell erzeugt werden. Das ist der Grundtyp eines synchronen Mikroprogramm-Steuerwerkes. Das Mikrobefehlsformat der IBM-Rechner schwankt zwischen 16 bit und 108 bit.

Zwischen den 8-bit-Typen 8080/8085 und der 16-bit-CPU 8086 wurde über ein Mikroprogramm-Steuerwerk im 86er auf diese Weise eine Dualität zwischen den Assembler-Mnemonics erreicht (beachte: Maschinenbefehlsquellcode auf Bitebene nicht gleich). Die 8086-CPU hat bekanntlich 29.000 Transistoren auf dem Chip, "einige" mehr als der Z 8000. Der 8086 hat einen Computer als Mikroprogramm-Steuerwerk auf dem Chip. Das erfordert nun mal einen höheren Aufwand an Transistoren. Bei der 6800-CPU ist das ebenfalls so. Ein Mikroprogramm-Steuerwerk ermöglicht ein leichtes Entwerfen eines speziellen Maschinenbefehlssatzes. Eine optimierte Random-Logik wie die des Z 8000 gestattet das nicht. Das ist der große kleine Unterschied.

Die Mikroprogramm-Befehlsdekodierung ist jedoch gegenüber einer optimierten Random-Logik relativ langsam und benötigt auch entsprechend Fläche auf dem Chip. Shima wird sich gedacht haben: "So mach ich das nicht, ich mach's ganz diskret". Diskret bedeutet die Entwicklung einer Logik nach den Bool'schen Gesetzen. Es bedeutet weniger Transistoren, weniger Gatter und damit geringere Signallaufzeiten als bei der Mikro-Dekodierung. Das ist der Grund, warum der Z 8000 eine so extrem schnelle Befehlsdekodierung besitzt.

Diese Random-Logik zu entwerfen, ist etwa mit der Leistung vergleichbar, als müßte man barfuß auf den Mount Everest klettern. Dieses Beispiel macht ein wenig klarer, wie dieser Shima einzuordnen ist. In der folgenden Zusammenstellung wird gezeigt, welchen zahlenmäßigen Aufwand an Transistoren er für die Dekodierung seiner drei Mikroprozessoren aufwendete:

Typ	8080 (LSI)	Z 80 (LSI)	Z 8000 (VLSI)
Technologie	NMOS (Gates auf hoher Spannung	NMOS (Niederspannungs-Verarmungstyp,	NMOS (Struktur reduziert)
Skalenfaktor einer Transistorstruktur (8080 = 100)	100%	70%	44%
Chip-Größe (mm^2	22,3	27,1	39,3
Absolute Transistorzahl	4'800	8'200	17'500
Anzahl der Transistoren für die Befehlskodierung mit festverdrahteter Logik	1'800	3'700	8'600
Chipfläche für Dekodierung in %	31	36	40
Zur Verfügung stehende Chipfläche in %	37	44	47
Anzahl der Maschinenbefehle	65	128	414

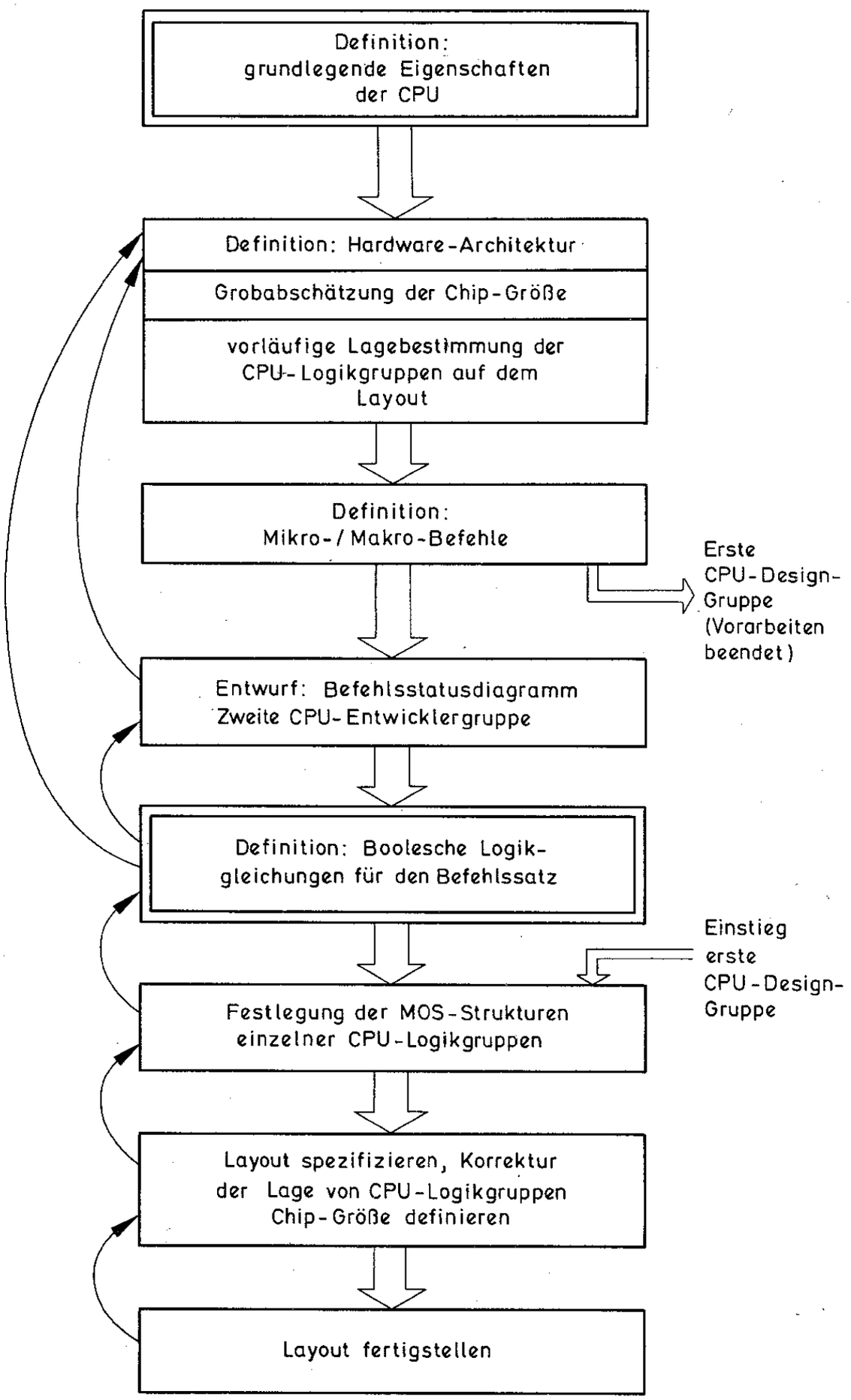

Definition:
grundlegende Eigenschaften
der CPU
Definition: Hardware-Architektur
Grobabschätzung der Chip-Größe
vorläufige Lagebestimmung der
CPU-Logikgruppen auf dem
Layout
Definition:
Mikro-/Makro-Befehle
Erste
CPU-Design-
Gruppe
(Vorarbeiten
beendet)
Entwurf: Befehlsstatusdiagramm
Zweite CPU-Entwicklergruppe
Definition: Boolesche Logik-
gleichungen für den Befehlssatz
Einstieg
erste
CPU-Design-
Gruppe
Festlegung der MOS-Strukturen
einzelner CPU-Logikgruppen
Layout spezifizieren, Korrektur
der Lage von CPU-Logikgruppen
Chip-Größe definieren
Layout fertigstellen

Bild 2.4

Der Skalenfaktor zwischen dem 8080 und dem Z 8000 liegt etwa bei 2:1. Geht man von der Struktur des Standard-NMOS-Prozesses aus, der etwa zwischen 4 und 6 Micron beträgt (4000 - 6000 μm^2 pro Zelle), liegt die Struktur des Z 8000 etwa bei 3 Micron. Dieses Maß kann man schon in den Bereich des "High density"-MOS-Prozesses einreihen. Zilog spricht nicht von dem HMOS-Prozeß. AMD spricht von einem 8000er Prozeß, der fast identisch mit dem der 8085-CPU ist. Man sollte jedoch bemerken, daß der Z 8000A eine noch kleinere Struktur als der 8000er aufweist. Der mit einem 6 MHz-Takt betriebene A-Typ ist teilweise mit der Geschwindigkeit von bipolaren Systemen vergleichbar, die ja bekanntlich extrem schnell sind.

Shimas Entwurf in Richtung hoher Datendurchsatz beschränkte sich jedoch nicht allein auf die Befehlsdekodierung. Die arithmetische 16-bit-Logikeinheit (ALU) weist eine 4-bit-Slice-Struktur auf, die, ebenfalls wie der Dekodierer, nach dem parallelen "carry look-ahead"-Prinzip arbeitet. Ferner hat Shima mehrere unabhängige Daten- und Steuerbusse zur Verknüpfung der internen Funktionsgruppen vorgesehen. Der primäre Befehlsdekoder ist redundant vorhanden. Shima nahm nach der Fertigstellung des Grundlayouts eine Feintrimmung des gesamten Z 8000-Logikkomplexes vor. Aus der Sicherheit genauester Detailkenntnisse nahm er Korrekturen vor, die sowohl die Laufzeit einzelner Pfade, wie auch das sichere Timing betrafen.

Das Resultat ist diese hochkarätige register-orientierte Architektur einer CPU, die in ihrer Art eine echte Ausnahmeerscheinung darstellt. Sie ist ein Meisterwerk menschlich erfaßbarer Logik. In **Bild 2.4** wird der Weg der Entwicklung des Z 8000 nochmals grafisch kurz zusammengefaßt. Die Pfeile deuten die Rückwirkung an, die bei den einzelnen Entwicklungsphasen entstanden. Der Z 8000 wurde nicht nach CAD-Techniken entworfen (Computer-aided-design). Es ist zwar phantastisch, wenn man eine künftige Architektur rein theoretisch simulieren kann, um ihre Leistungsfähigkeit zu erfassen. Man darf jedoch den sehr hohen Aufwand bei dieser Prozedur nicht vergessen. Der Simulationscomputer erfordert eine entsprechend lange Programmierung. Außerdem ist nicht sicher gestellt, daß kritische Laufzeitstrecken auch mit der notwendigen Genauigkeit zu simulieren sind; ganz einfach deshalb, weil man einzelne Parameter zu wenig genau definieren kann.

Bild 2.5 zeigt den vereinfachten inneren Aufbau der Z 8000-CPU. Der primäre "look-ahead"-Befehlsdekoder gibt Operanden irgendeines Allzweckregisters an die ALU frei. Hier zeigt sich auch, daß die Allzweckregister keine Pseudo-Akkus sind, sondern echte. Das Register, das nach der Befehl-Mnemonic zuerst bezeichnet ist, enthält nach der Operation auch das Ergebnis der ALU.

Von dem Block links unten mit der Bezeichnung CPU-Steuerregister wird die Adreß-/Daten-Multiplexlogik umgeschaltet. Die CPU überträgt an ihren AD-Anschlüssen sowohl Adressen (nur nach außen) und Daten (in beiden Richtungen). Diese primären CPU-Signale werden bei 16-bit-CPUs zunehmend auf einen gemeinsamen Bus gelegt. Wie schon angedeutet, liegt der Hauptgrund darin, Groß-DIL-Gehäuse zu vermeiden. Gegen diese Technik ist im Grunde garnichts zu sagen. Das Zeitverhalten zwischen Adresse und Daten ist absolut eindeutig definiert, wie in einem späteren Abschnitt noch dargestellt wird. Ein weiterer, sehr wesentlicher Grund ist, daß die Adresse und die Daten intern niemals gleichzeitig ausgegeben werden. Ein Mikroprozessor ist eine taktgesteuerte Logik. Zuerst wird der Programmzähler, ein Adreßregister oder der Stackinhalt (Adresse) ausgegeben und frühestens zwei Takte später Daten aus einem Register hinausgegeben bzw. hereingeholt. Auch ein 68000 macht das so. Überdies hat jeder Speicher eine endliche Zugriffszeit. Ist sie mit z.B. 150 ns angegeben, so ist erst nach die-

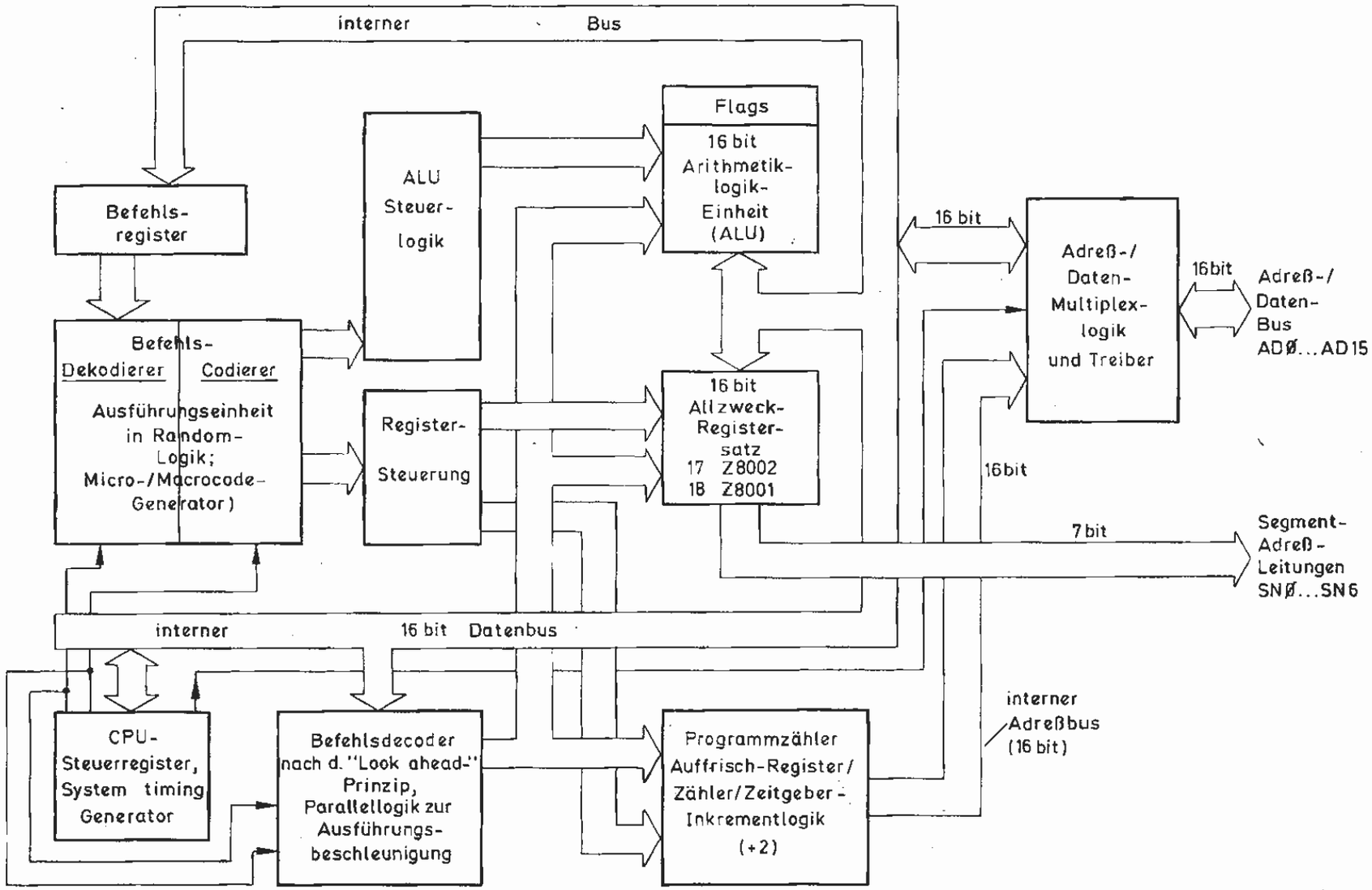

interner Bus
Befehls-register
ALU Steuer-logik
Flags
16 bit Arithmetik-logik-Einheit (ALU)
16 bit
Adreß-/ Daten-Multiplex-logik und Treiber
16bit
Adreß-/ Daten-Bus AD0...AD15
Befehls-Dekodierer Codierer
Ausführungseinheit in Random-Logik; Micro-/Macrocode-Generator)
Register-Steuerung
16 bit Allzweck-Register-satz
17 Z8002
18 Z8001
16bit
7bit
Segment-Adreß-Leitungen SN0...SN6
interner 16 bit Datenbus
CPU-Steuerregister, System timing Generator
Befehlsdecoder nach d. "Look ahead-" Prinzip, Parallellogik zur Ausführungs-beschleunigung
Programmzähler Auffrisch-Register/ Zähler/Zeitgeber-Inkrementlogik (+2)
interner Adreßbus (16 bit)

Bild 2.5

ser Zeit der Transfer aktiv. Es würde nichts bringen, wären Adresse und Daten an getrennten CPU-Pins zur gleichen Taktflanke aktiv. Gemeinsame Adreß-/Daten-Anschlüsse am CPU-Baustein sind kein Rückschritt. Schon 1975 wurde diese Bus-Struktur erstmals verwendet. Der 12-bit-Prozessor IM(HM) 6100 hat diese Doppelbelegung.

Zusammenfassend kann man festhalten, daß bei der Entwicklung der Z 8000-CPU durchweg alle schaltungstechnischen Prinzipien in die Tat umgesetzt wurden, die einen sehr schnellen Datendurchsatz sichern. Gemeinsame Anschlüsse für Adresse und Daten sind eine konsequente Maßnahme, um die Gehäusegrößen von 16-bit-CPUs denen von 8-bit-Typen anzugleichen.

2.3 SEGMENTIERTE UND UNSEGMENTIERTE VERSION

Der Index 8000 hat einen Grund. Die CPU wurde dafür konzipiert, insgesamt 128 Segmente mit einer maximalen Größe von 64 KB direkt adressieren zu können. Das bedeutet eine Menge adressierbarer Bytes von 128 x 65'536 Bytes = 8'388'608 Bytes. Das sind über den Daumen abgerundet 8000 KByte. Setzt man noch ein Z davor und läßt die Benennung weg, ist alles klar. Etwas schwieriger wird es dann, wenn man die Endadresse des letzten Segments bestimmen will. Sie ergibt sich aus 127 x 65'535 → 7FH x FFFFH = 7EFF81H = 8'322'945.

Der Z 8000 existiert in zwei Versionen. Der eigentlich typischen Z 8000-CPU hat man die Bezeichnung Z 8001 gegeben. Man nennt sie auch "die Segmentierte". Dieser Typ hat einen kleineren Bruder, den Z 8002. Ihn nennt man "den Unsegmentierten". Das Bild im Vorwort am Anfang des Buches zeigt beide einträchtig beisammen, den 40poligen Baustein Z 8002 und den 48poligen Z 8001.

Als Chip existiert nur ein Typ. Der etwa 40 mm^2 große Chip hat 49 Kontaktierungspunkte. Er wird wahlweise nach einem Testergebnis zu einer 8001- oder einer 8002-CPU verwendet. **Bild 2.6** zeigt eine Fotografie dieses Superchips. Es ist nicht ganz einfach, an Hand dieses Fotos die Lage der Funktionsgruppen deutlich zu zeigen. Das große, etwas unregelmäßige Feld in der Mitte ist die Befehls-Dekodierlogik. Oberhalb, etwas rechts von der Mitte liegt das Feld der Allzweck-Register; oberhalb, etwas links, die 16-bit-ALU und ganz links oberhalb die Flag- und Bedienungslogik. Damit sind die wichtigsten Funktionsgruppen genannt.

Man kann sich vorstellen, daß zwischen dem 8001 und dem 8002 im Grunde nicht viel Unterschied sein kann. Das ist auch in der Tat so. Die Unterschiede liegen im Adressierungsbereich, den Stapelzeiger-Registern und den Programmstatus- und -Bereichszeiger-Registern. In **Bild 2.7** ist der rein hardwaremäßige Unterschied zwischen beiden CPU-Typen dargestellt. Die zusätzlichen acht Anschlüsse ergeben sich aus den sieben Segmentleitungen für die Adresse und dem Segment-Trap-Eingang, der von dem MMU-Baustein herangeführt wird. Generell ist an Hand des Bildes 2.7 festzustellen, daß der Z 8000 eine klare, leicht verständliche Signalgruppierung aufweist. Die Gruppen sind mit Zeitsteuersignale, Betriebszustand, CPU-Steuerung, Bussteuerung, Interrupt-Eingänge, Multiprozessor-Interface und Adreß-/Daten-Bus zu überschreiben.

Der Z 8002 arbeitet nur in einem 64 KB Adreßbereich. Der Z 8001 kann in zwei Adreßbereichen arbeiten, d.h. segmentiert im 8 MB Adreßbereich oder nichtsegmentiert im 64 KB Adreßbereich. Definiert wird die Betriebsart im Flag- und Steuerwort-Register (FCW). Das

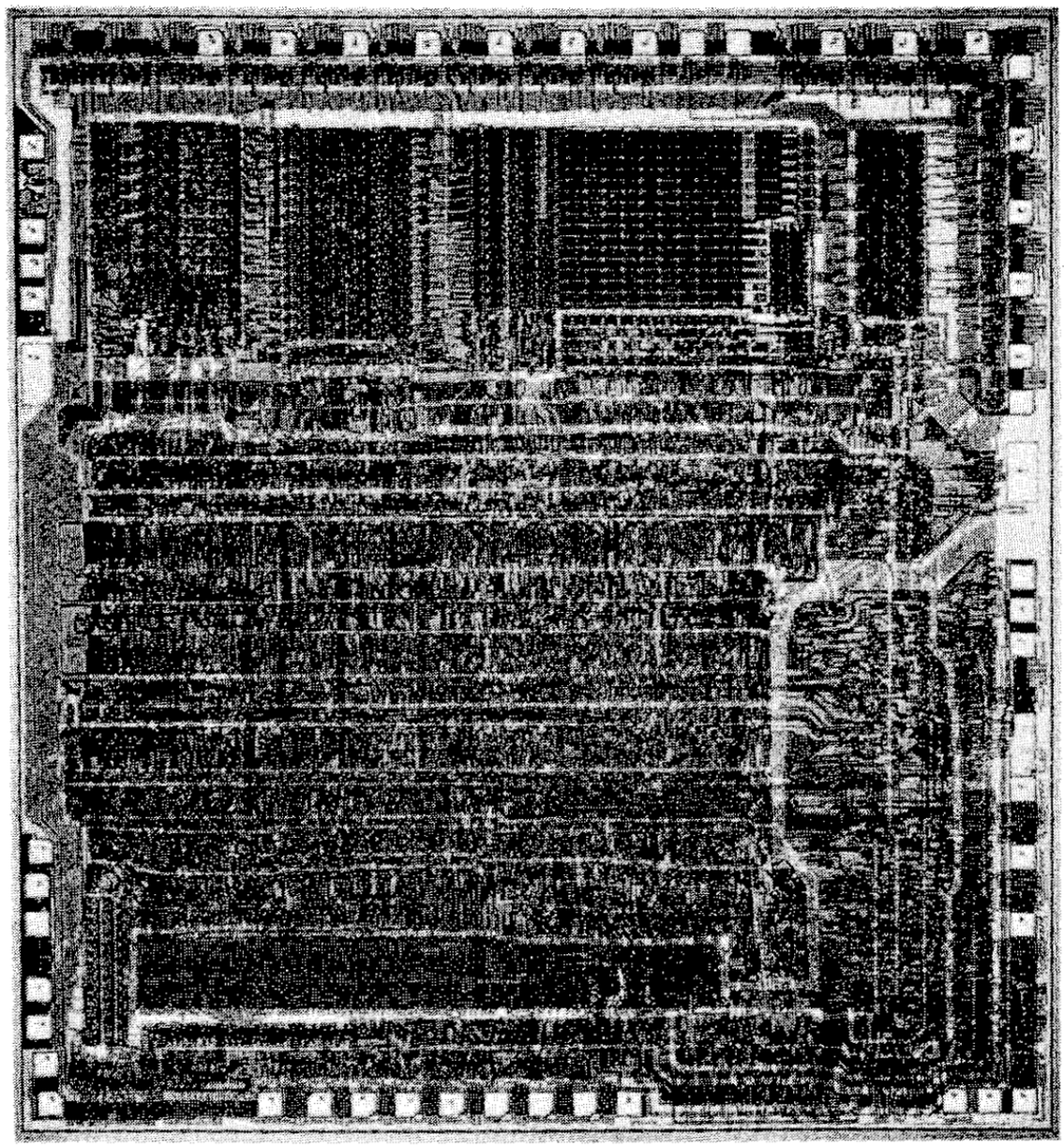


Bild 2.6

FCW-Register beinhaltet in bit 15 die Bezeichnung SEG. Ist das SEG-bit gleich 1 (Hi), so ist der Z 8001 im segmentierten Modus (0 entsprechend nichtsegmentiert). Im FCW-Register des Z 8002 ist bit 15 immer Null. Ein Z 8001 im nichtsegmentierten Modus bearbeitet Programme wie der Z 8002. Programme im nichtsegmentierten Modus enthalten entweder keine Segmentadresse, oder die CPU beachtet den Segmentteil nicht. Das äußert sich beispielsweise darin, daß bei dem Befehl eines Unterprogrammaufrufes, genannt CALL "Procedure", das Stapelzeiger-Register im nichtsegmentierten Modus nur um zwei Stellen (z.B. 0022H auf 0020H) statt um vier Stellen im segmentierten Modus zurückgesetzt wird. Eine komplette Segmentadresse benötigt 4 Bytes, zwei für das Segment und zwei für den Offset. Bei einem Rücksprung aus einem Unterprogramm (UPRO) ist es genauso. Mit dem Rücksprungbefehl RET "Return from Procedure" werden dem Stapelzeiger im nichtsegmentierten Modus nur zwei Stellen hinzugezählt (z.B. 0020H auf 0022H), statt vier Stellen im segmentierten Modus. Die grafische Beschreibung eines beliebigen Segments ($0 \leqslant n \leqslant 127$) wird in **Bild 2.8** gezeigt. Die logische Adresse des Z 8001 besteht aus Segmentteil (7 bit) und Offsetteil (16 bit). Die Segmentanfangsadresse bedeutet den Wert des Segmentteils, vorangeschrieben den 16 Nullen für Offset gleich Null.

Offset hört man auch bei Operationsverstärkern. Es besteht zwischen digitalem und analogem Offset wohl ein Zusam-

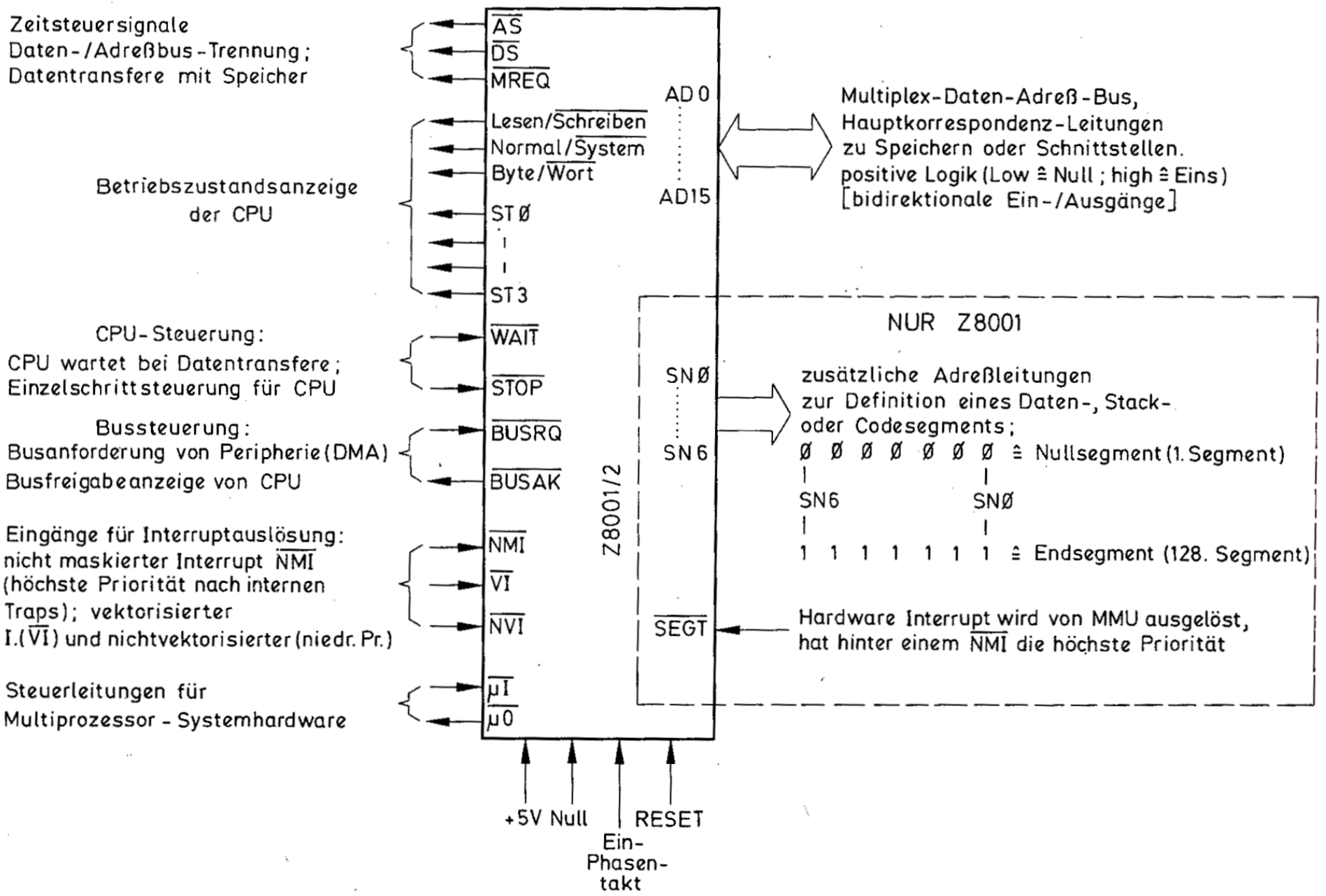

Zeitsteuersignale
Daten-/Adreßbus-Trennung;
Datentransfere mit Speicher
AS
DS
MREQ
Betriebszustandsanzeige
der CPU
Lesen/Schreiben
Normal/System
Byte/Wort
ST Ø
ST 3
AD 0
AD15
Multiplex-Daten-Adreß-Bus,
Hauptkorrespondenz-Leitungen
zu Speichern oder Schnittstellen.
positive Logik (Low ≙ Null ; high ≙ Eins)
[bidirektionale Ein-/Ausgänge]
NUR Z8001
CPU-Steuerung:
CPU wartet bei Datentransfere;
Einzelschrittsteuerung für CPU
WAIT
STOP
SN Ø
SN 6
zusätzliche Adreßleitungen
zur Definition eines Daten-, Stack-
oder Codesegments;
Ø Ø Ø Ø Ø Ø Ø ≙ Nullsegment (1. Segment)
SN6 SNØ
1 1 1 1 1 1 1 ≙ Endsegment (128. Segment)
Bussteuerung:
Busanforderung von Peripherie (DMA)
Busfreigabeanzeige von CPU
BUSRQ
BUSAK
Z8001/2
Eingänge für Interruptauslösung:
nicht maskierter Interrupt NMI
(höchste Priorität nach internen
Traps); vektorisierter
I.(VI) und nichtvektorisierter (niedr. Pr.)
NMI
VI
NVI
SEGT
Hardware Interrupt wird von MMU ausgelöst,
hat hinter einem NMI die höchste Priorität
Steuerleitungen für
Multiprozessor - Systemhardware
µI
µO
+5V
Null
RESET
Ein-
Phasen-
takt

Bild 2.7

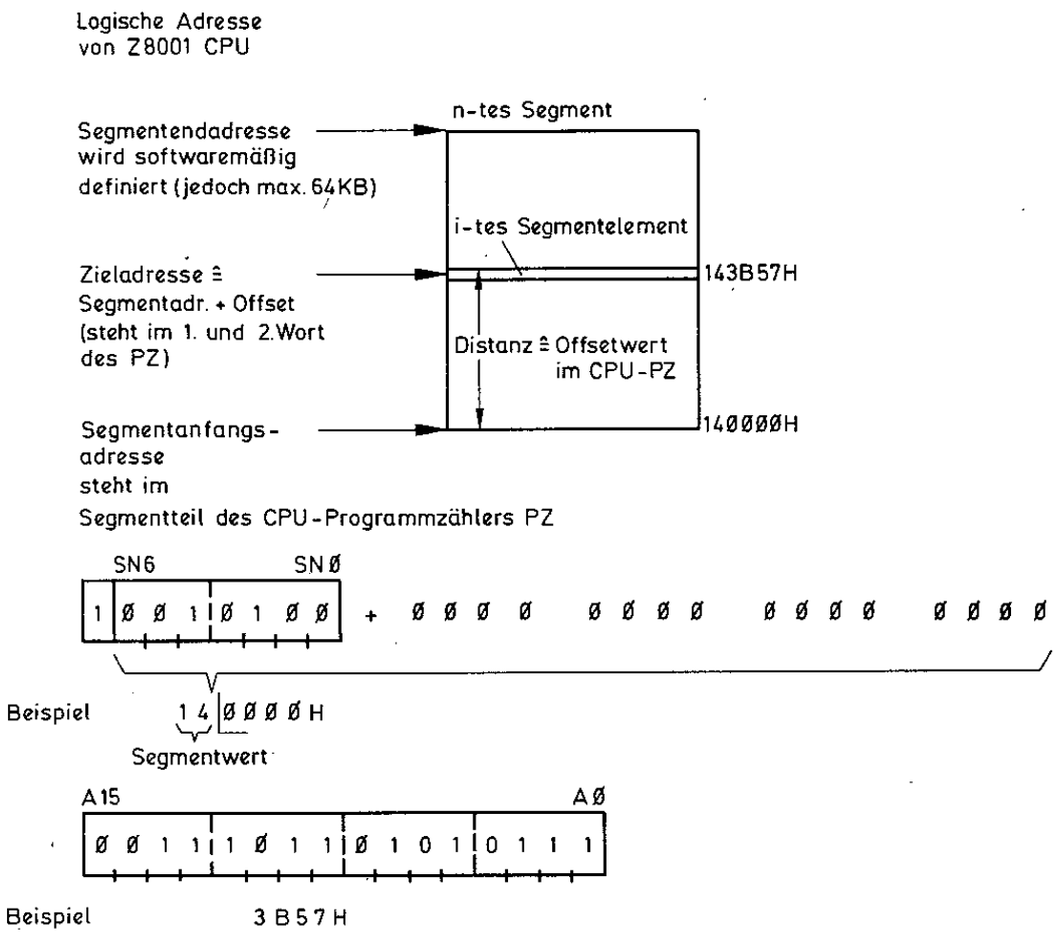


Bild 2.8

menhang. Man nehme an, die Adresse 140000H entspricht einem bestimmten Spannungspegel. Die Offsetadresse kann nun eine Schwankung dieses Pegels um 1/128 seines Wertes bewirken, wenn sich die Adresse von 0000H auf FFFFH ändert.

Die Zieladresse in einem Segment ist die Summe aus dem Segmentteil + Offset; in dem gezeichneten Fall also 143B57H. Die Befehle CALL und RET benützt man im segmentierten Modus, wenn z.B. das Hauptprogramm in Segment 16 (10H) und das UPRO in Segment 20 (14H) steht. Über CALL "UPRO Seg. 20, Offset 1234H" wird gesprungen und RET "main prog., Offset X + 2" wird zurückgekehrt. Ein Beispiel für UPRO-Aufrufe wird in Kapitel 6 erläutert.

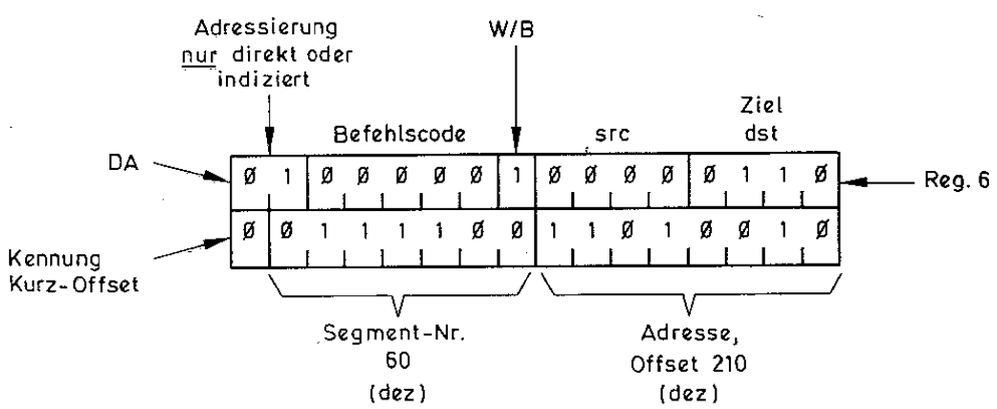
Bei der Z 8001-CPU wird in Form einer Programmzählervariation unterschieden, wie groß ein Segment ist. Es gibt genügend Fälle, in denen ein Segment nicht größer als 256 Byte zu sein braucht. Man spricht hier in diesem Zusammenhang von einem Segment mit einem kurzen Offset. Ein kurzer Offset ist eine Adresse, die nur in einer 8-bit-Länge zu beschreiben ist. Das höherwertige Byte der Adresse ist immer Null. Will man eine Zieladresse in einem Segment dieser Art definieren, benötigt man nur ein 16-bit-Adreßwort. Die Beschreibung eines kurzen Offset mit Nullen im höherwertigen Byte ist ähnlich zu der Zero-Page-Adressierung des 8-bit-Prozessors des Typs 6502. Der 6502 wie der Z 8001 zählt die Nullen für das obere Byte "gedanklich" dazu. Für die Beschreibung eines Kurz-Offset-Segments werden nur zwei der acht Adressierungsarten des Z 8000 verwendet. Für die richtige syntaktische Behandlung verwendet man die direkte Adressierung (Operand ist Inhalt einer Adresse im Befehl) und die indirekte (indiziert) Adressierung (Adresse im Befehl plus Differenzadresse in Register).

Im Assembly Language Programming Manual findet man unter der Beschreibung der Befehle die Abkürzung "SS", die die Segmentadressierung eines Kurz-Offset-Segments andeutet. Ein Beispiel wird in **Bild 2.9** gezeigt. Die Additionsoperation von einer Kurz-Segment-Speicherstelle in einen Allzweck-Register ist eine 4-Byte-Operation. Die symbolische Beschreibung eines Kurz-Offset-Segments lautet demnach folgendermaßen:

R 0 ≪ 60 ≫ 210

Register Segment-Nr. Adresse (dez)

Im Register R 0 könnte z.B. die aus Bild 2.9 verwendete Kurz-Segment-Adresse als Dedezimalwert 3CD2H stehen. Dazu muß nachdrücklich angemerkt werden, daß dieses Darstellung schon sehr symbolisch gemeint ist, denn eine bloße Adreßangabe für eine Prozedur bzw. ein UPRO ist sehr selten. Der Aufruf CALL "Procedure" besagt, daß dem UPRO ein Name zugeordnet ist, unter dem es auch aufgerufen wird. Das UPRO wird überwiegend in einer Link-Operation in das Speichersystem eingegliedert. Aber trotzdem, ein UPRO wird manchmal nur unter Angabe der Adresse aufgerufen.



Addiere Inhalt Speicherstelle 210 von Segment 60 in den Register 6

ADD R6, ≪60≫ % ØØD2

Bild 2.9

In **Bild 2.10** werden beide Fälle eines Segmenttyps dargestellt, wie sie vom Z 8001-Programmzähler behandelt werden. Das MSB (most significant bit) ist maßgebend, ob es sich um ein kurzes Segment (bis 256 Byte) oder langes Segment (bis 64 KB) handelt. Damit wird auch definiert, ob die komplette Segmentadresse (kombiniert Segment und Offset) als 2-Byte-Programmzählerelement oder aus 4 Bytes bzw. 2 Worten zu verstehen ist. Bei großen Segmenten ist das niederwertige Byte des Wortes mit dem 7-bit-Segmentteil unwirksam.

Eine Bemerkung sei zu dem Komplex Segmentierung noch angefügt. Nach jedem ausgeführten Befehl wird der Programmzähler entsprechend inkrementiert, bis sein Inhalt auf den nächsten Befehl im Programm zeigt (falls kein Interrupt oder Sprung einen anderen PC-Wert vorgeben). Erreicht der Programmzähler den ominösen Wert FFFFH, so erfolgt nach einem weiteren Inkrementieren **keinesfalls** ein Sprung in das nächst höhere Segment.

Auch die Segmentadresse wird nicht inkrementiert. Es erfolgt kein Übertrag der Offsetadresse in die Segmentadresse. Das wäre auch grober Unfug, denn das ganze Gerede über die Segmentbehandlung wäre "nonsense". Spätestens zum Programmzählerstand FFFFH muß ein Sprung erfolgen, oder die CPU generiert einen Trap. Das gilt genauso für ein Kurz-Offset-Segment, wenn der Pro-

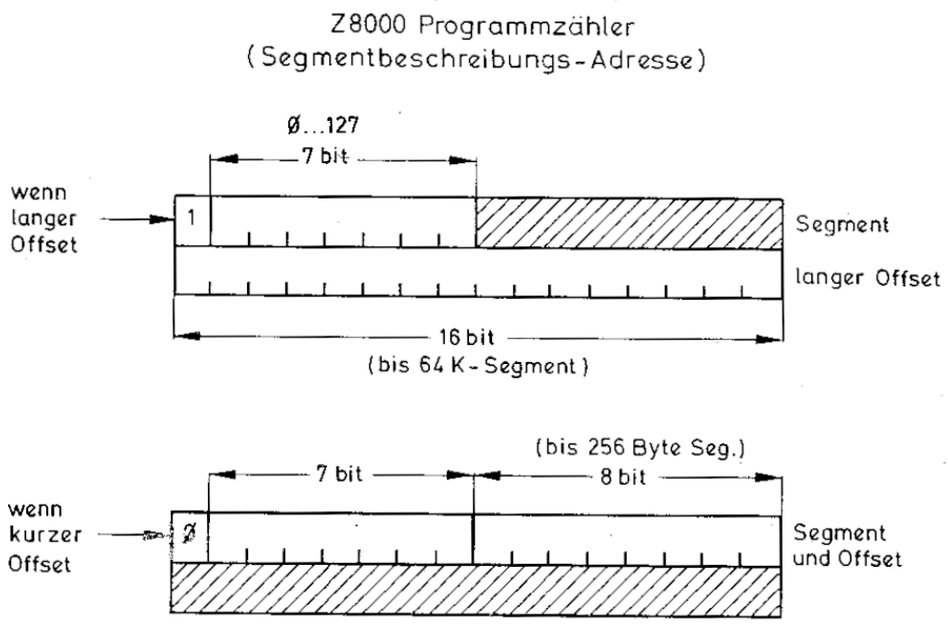


Bild 2.10

grammzähler den Wert 255 (FFH) überschreitet. Einen anderen legalen Segmentübertritt als mit einem CALL, RET oder IRET gibt es nicht.

2.4 ANSCHLUSSBILDER UND PINBESCHREIBUNG BEIDER VERSIONEN

Dieser Abschnitt ist ausschließlich den Anschlußbildern und -bezeichnungen beider Z 8000-Versionen gewidment.

Wie bereits aus Bild 2.7 ersichtlich war, ist der Z 8000-Chip mit einer +5 V-Spannungsquelle zu betreiben. Bei dieser Spannung benötigt der Chip maximal 300 mA Versorgungsstrom. Die dabei konstant aufgenommene Leistung von etwa 1,5 W ergibt bei längerer Betriebszeit eine Bausteintemperatur von 50 bis 60 °C. Wird eine Rechnerplatine in einem schwach belüfteten Gehäuse betrieben, wird die Bausteintemperatur bei Umgebungstemperaturen von 25 bis 30 °C erfahrungsgemäß weiter steigen. Bis +70 °C Bausteintemperatur wird die sichere Funktion garantiert.

Die überwiegende Zahl aller Steuersignal-Ein-/Ausgänge sind Low-aktiv (ausgenommen der AD-Bus, vier Statusausgänge und die Segmentadreßausgänge). Ist demzufolge der Eingangs- bzw. Ausgangspegel geringer als +0,6 V, sind die entsprechenden Anschlüsse ihrer Bedeutung nach aktiv.

Eine weitere Gemeinsamkeit der Mehrzahl der CPU-Anschlüsse ist die Tri-State-Ausgangsschaltung. Diese Eigenschaft haben der AD-Bus, die Segmentadreß-Ausgänge, die Zeitsteuer- und Betriebszustands-Ausgänge gemeinsam. Ein Tri-State-Ausgang gilt generell als vollkommen inaktiv, wenn er im hochohmigen Zustand ist. Dieser Zustand ist mit einer Impedanz von 100 K bis 500 KΩ erreicht. Es fließen über den Ausgang noch Leckströme im "Aus"-Zustand von 50 bis 10 μA. Bei CMOS-Schaltungen können Impedanzen bis 2 MΩ im "Aus"-Zustand erreicht werden. Für den Z 8000 gelten in diesem Zustand Impedanzen von etwa 500 KΩ.

Im Gegensatz zur Z 80-CPU fehlt beim Z 8000 der $\overline{\text{RFSH}}$-Anschluß. Dies ist beim Z 8000 nicht notwendig. In einem

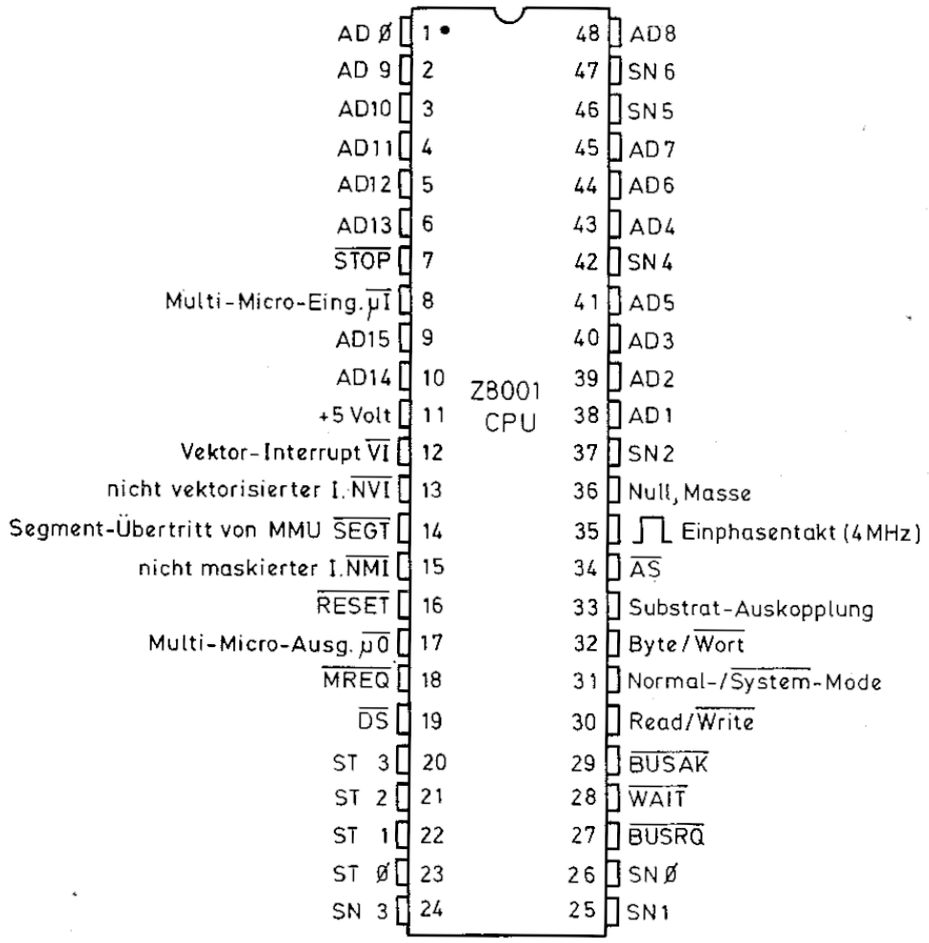


Bild 2.11

internen Refresh-Steuerregister ist der chronologische Ablauf einer Auffrisch-Adreßausgabe für dynamische RAM-Speicher programmierbar. Die Auffrisch—Adresse wird wie jede andere Adreßausgabe durch $\overline{AS}$ gleich Low angezeigt.

Bild 2.11 und **2.12** zeigen die Anschlußbilder der beiden Z 8000-Versionen. In der folgenden Auflistung der Bausteinanschlüsse wird ihre Funktion beschrieben. Falls bei dieser Beschreibung Zeitangaben für Signale gemacht werden, beziehen sie sich grundsätzlich auf eine Taktzykluszeit von 250 ns.

AD 0 . . . AD 15 (Adressen, Daten):

Diese 16 Anschlüsse übertragen den primären Informationsaustausch zwischen der Peripherie (Speicher oder E/A-Port) und der CPU. Jede Leitung hat drei Funktionen: Adreßbit ausgeben, Datenbit übernehmen (lesen) oder ausgeben (schreiben). Die Adreßausgabe ist ein binäres Wort mit der höchsten Stelle A 15 (MSB) und der niedrigsten Stelle A 0 (LSB). Das Datenwort muß nicht ausschließlich binär verknüpft sein, ist es jedoch überwiegend. Für Daten und Adressen gilt gleichermaßen Low (Lo) gleich Null und High (Hi) gleich Eins. Der Zustand der Zeitsteuersignale $\overline{AS}$ und $\overline{DS}$ gibt eindeutig an, wann an den AD-Anschlüssen ein Adreß- oder Datenwort anliegt.

$\overline{AS}$ Adreß-Strobe (Ausgang):

Dieser Anschluß ist immer im ersten Taktzyklus eines CPU-Maschinenzyklusses etwa 120 ns aktiv. Er zeigt damit das Anliegen eines Adreßwortes am AD-Bus an. Die Adresse ist bei $\overline{AS}$ gleich Lo in der Phase der Ausgabe und im weiteren

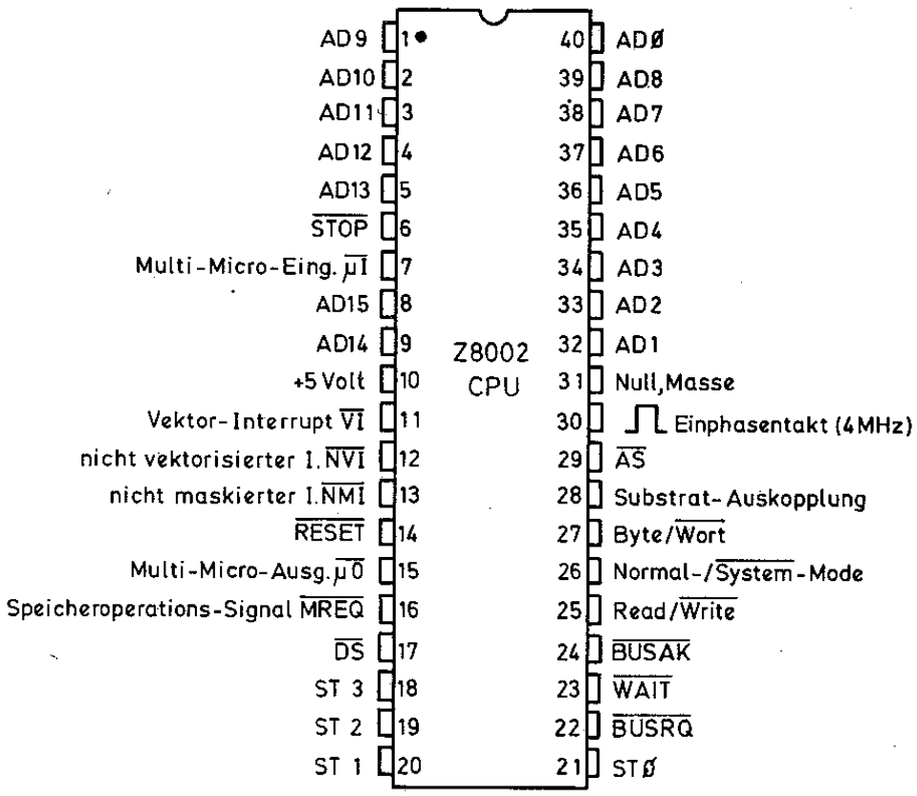


Bild 2.12

Verlauf vorhanden. Sie wird jedoch bei der Lo↑Hi-Flanke erst als absolut gültig definiert.

$\overline{DS}$ Daten-Strobe (Ausgang):

Dieser Anschluß wird frühestens im zweiten Taktzyklus eines CPU-Maschinenzyklusses aktiv; frühestens nach 370 ns gemessen von der Lo↑Hi-Taktflanke. $\overline{DS}$ gleich Lo zeigt an, daß von diesem Zeitpunkt an Daten von der CPU akzeptiert (lesen) oder von ihr ausgesandt werden (schreiben). Für $\overline{DS}$ aktiv entscheiden unterschiedliche Operationen: Speicher lesen (370 ns), Speicher schreiben (470 ns) und E/A-Port lesen/schreiben (500 ns). $\overline{DS}$ ist ein reines CPU-Zeitstatussignal. $\overline{DS}$ bleibt länger aktiv, wenn über den $\overline{WAIT}$-Eingang der CPU Wartezyklen eingeschoben werden, d.h. der Datentransfer wird künstlich verlängert. Über den R/$\overline{W}$-Ausgang wird die Richtung des Datentransfers von der CPU definiert.

$\overline{MREQ}$ Memory Request (Ausgang):

Dieser Ausgang wird im Lo-Abschnitt des ersten Taktzyklusses eines CPU-Mz wirksam. Er definiert genau, um welche Art eines Datentransfers es sich bei dem momentanen **M**aschinen**z**yklus handelt. Lo zeigt einen Speichertransfer an. Hi zeigt einen Transfer mit einem E/A-Port an.

R/$\overline{W}$ lesen/$\overline{schreiben}$ (Ausgang):

Der Zustand dieses Ausgangs definiert die Richtung eines Datentransfers. Hi zeigt an, daß das Datenbyte/-wort von der Peripherie in die CPU übertragen wird (lesen). Lo zeigt an, daß die CPU an die Peripherie ein Datenbyte/-wort ausgibt (schreiben). Der Ausgang wird im Hi-Abschnitt des ersten Taktzyklusses eines CPU-Mz wirksam.

N/$\overline{S}$ Normal/$\overline{System}$ (Ausgang):

Die Z 8000-CPU besitzt zwei Betriebs-

arten, die sich in der Ausführbarkeit aller Z 8000-Befehle unterscheiden. Die insgesamt 110 Befehlstypen der CPU enthalten einige bevorzugte Befehle, die nur dann ausgeführt werden können, wenn die Betriebsart "System" gewählt ist (im FCW-Bit 14 setzen). Hi zeigt die **Normal**-Betriebsart an, die die Ausführung bevorzugter Befehle **nicht** gestattet. Lo zeigt die **System**-Betriebsart an, die die Ausführung aller Befehle gestattet.

B/$\overline{W}$ Byte/$\overline{\text{Wort}}$ (Ausgang):

Dieser Ausgang wird im Hi-Abschnitt des ersten Taktzyklusses eines CPU-Mz wirksam. Sein Zustand definiert einen Datentransfer um einen weiteren Grad. Der Z 8000 führt Byte- und Wortbefehle (plus Langwortbefehle) aus. Dieser Ausgang zeigt den Unterschied eines Byte-Maschinenzyklusses (gleichbedeutend mit Hi) oder eines Wort-Maschinenzyklusses (Lo) an. Speziell bei Speicherdaten-Transfers wird zur hardwaremäßigen Unterscheidung dieser Zyklen die Adreßleitung A 0 und B/$\overline{W}$ benützt. Ist A 0 gleich Lo (= gerade Adresse) wird für B/$\overline{W}$ gleich Hi das höherwertige Daten-Byte (D 8 bis D 15) beachtet.

ST 0 bis ST 3 Status (Ausgänge):

Diese vier Anschlüsse sind binär verknüpft und ergeben entsprechend 16 diskrete Zustände. 15 davon definieren einen momentanen Arbeitsgang (Maschinenzyklus) der CPU. Diese Anschlüsse zeigen den hohen Stand der interaktiven Arbeitsweise der CPU. Sie zeigen genau an, welchen Operationstyp die CPU gerade ausführt. Dekodiert man die Signale in diskrete Zustände, können sie hardwaremäßig weiter verwendet werden. Die ST-Ausgänge sind im Hi-Abschnitt des ersten Taktzyklusses eines CPU-Mz wirksam.

$\overline{\text{WAIT}}$ CPU-Steuerung (Eingang):

Über diesen Eingang kann man die CPU dazu veranlassen, einen Datentransfer zeitlich auf die Peripherie abzustimmen. Die Peripherie müßte bei einem Transfer innerhalb von 350 ns reagieren können (bzw. etwa 250 ns bei der Einschaltung einer MMU als Adreßgenerator). Diese Zeiten sind für einen überwiegenden Teil der Speicherbausteine und E/A-Ports zu schnell. Ist während einer Hi↓Lo-Flanke im zweiten Taktzyklus eines CPU-Mz ein Lo am WAIT-Eingang anliegend, so schiebt die CPU einen Wartezyklus (WZ) bei einem Speichertransfer ein. Ein WZ entspricht der Zeit von 250 ns. Der AD-Bus und alle wichtigen Status- und Zeit-Signale der CPU bleiben in dieser Zeit unverändert. Bei einem E/A-Transfer werden zu einem automatisch eingeschobenen WZ ein weiterer hinzugefügt, wenn bei der Hi↓ Lo-Flanke des WZ-Taktes am WAIT-Eingang Lo anliegt. WZ's können beliebig eingeschoben werden.

$\overline{\text{STOP}}$ CPU-Steuerung (Eingang):

Der STOP-Eingang ermöglicht eine einfache Kontrolle über die Ausführung eines Programms. Man kann dabei immer nur einen Maschinenzyklus von der CPU ausführen lassen. Ist der STOP-Eingang aktiviert, wird die CPU angehalten. Sie gibt im angehaltenen Zustand nur noch Speicherauffrischadressen aus. Der STOP-Eingang wird im letzten Takt eines CPU-Mz abgefragt. Ist STOP gleich Lo, wird nur noch der folgende Befehl geholt. Der STOP-Eingang wird auch bei der Hardware-Verknüpfung mit dem Entlastungsprozessor EPU benutzt.

$\overline{\text{BUSRQ}}$ Bus-Anforderung (Eingang):

Wird der CPU-Bus von einem DMA-Baustein (z.B. Z 8016 DMATC) benötigt, um einen Datentransfer im direkten Speicherzugriffs-Modus vorzunehmen, muß die Anmeldung für die Bussteuerung über den BUSRQ-Eingang eingeleitet werden. Wird Lo an diesen Anschluß gelegt, wird in der CPU ein Flipflop gesetzt. Nach dem dritten Takt eines beliebigen Mz's werden der AD-Bus, die Segmentadreß-Ausgänge, die Zeitsteuer- und Betriebszustands-Ausgänge (nach Bild 2.7) in den hochohmigen Zustand

versetzt. Gleichzeitig antwortet die CPU mit Lo am Anerkennungsausgang. Je nach Dauer der DMA-Operation muß BUSRQ auf Lo gehalten werden und dieser wird anschließend von dem DMA-Baustein selbst wieder inaktiviert. Nach zwei Taktzyklen übernimmt die CPU wieder die Busprioritat.

$\overline{\text{BUSAK}}$ Busabgabe-Anerkennung (Ausgang):

Ist der BUSRQ-Eingang der CPU aktiviert worden, wird die Anforderung auf die Busprioritat eines DMA-Bausteins von der CPU nach etwa drei Taktzyklen anerkannt. Die Busabgabe-Anerkennung der CPU erfolgt durch Lo am Anschluß BUSAK.

$\overline{\text{NMI}}$ nicht maskierter Interrupt (Eingang):

Dieser Programmunterbrechungs-Eingang besitzt die höchste Priorität aller Interruptebenen des Z 8000. Er folgt in der generellen Priorität nach einem internen Trap. Er wird flankengesteuert (Hi↓Lo-Flanke). Ein nur für diese Ebene vorhandenes Flipflop aktiviert den NMI zu jeder Zeit des Mz.

$\overline{\text{VI}}$ vektorisierter Interrupt (Eingang):

Die nächst niedrigere Priorität der Programmunterbrechung über Interrupts hat der VI. Er folgt in der generellen Priorität nach einem Segment-Trap der MMU. Der VI muß so lange angelegt sein (Lo), bis eine Anerkennung der CPU durch ein entsprechendes Statussignal erfolgt. Ein VI wird niemals anerkannt, wenn das Bit 12 im FCW, genannt VIE, nicht gesetzt ist.

$\overline{\text{NVI}}$ nicht vektorisierter Interrupt (Eingang):

Die absolut niedrigste Priorität einer Unterbrechung repräsentiert der NVI. Er muß so lange angelegt sein (Lo), bis eine Anerkennung der CPU durch ein entsprechendes Statussignal erfolgt. Ein NVI wird niemals anerkannt, wenn das Bit 11 im FCW, genannt NVIE, nicht gesetzt ist.

$\overline{\mu\text{I}}$ Multi-Mikro (Eingang):

Über diesen Eingang wird eine Anforderung auf die Busprioritat in die CPU eingegeben. Dieser Eingang ist Teil der Multi-Mikro Daisy-Chain-Buskonfiguration eines Mehrprozessorsystems.

$\overline{\mu\text{O}}$ Multi-Mikro (Ausgang):

Über diesen Ausgang wird eine Anforderung auf die Busprioritat von der CPU ausgegeben. Dieser Ausgang ist Teil der Multi-Mikro Daisy-Chain-Buskonfiguration eines Mehrprozessorsystems.

$\overline{\text{RESET}}$ (Eingang):

Wird Lo an den RESET-Eingang gelegt, werden zuerst alle Funktionsleitungen der CPU in einen inaktiven Zustand gebracht. Nach drei Taktzyklen beginnen zwei Hohlzyklen bei der Z 8002-CPU für das Laden des FCW-Wertes und der Startadresse für den Programmzähler. Die Z 8001-CPU erzeugt drei Hohlzyklen: FCW, Programmzähler-Segment, und -Offset.

$\overline{\text{CLK}}$ CPU-Takt (Eingang):

Die Z 8000-CPU ermöglicht den Betrieb mit einem Einphasentakt in einem Bereich von 500 kHz bis 4 MHz (entsprechend 2 μs . . . 250 ns Zykluszeit). An diesen Takt wird die Anforderung gestellt, daß der Hi-Lo-Abschnitt möglichst synchron ist (105 ns sollen nicht unterschritten werden). Die Flanken sollen in 20 ns oder schneller fallen oder steigen. Ausgenommen ist der Z 8000A, der mit 6 MHz arbeiten kann (bzw. 166 ns Zykluszeit). Keinesfalls darf mit einem statischen Pegel am CLK-Eingang gearbeitet werden. Auf die Dauer führt das unweigerlich zu einer Zerstörung des Chips. Der Hi-Pegel sollte $U_C - 0{,}4$ V (= 4,6 V) nicht unterschreiten. Der aus einem TTL-Oszillator (74S04) erzeugte Takt sollte über eine Komplementär-Treiberschaltung den Anforderungen an MOS-Bausteinen angepaßt werden. **Bild 2.13** zeigt eine Variante eines exakt vor-

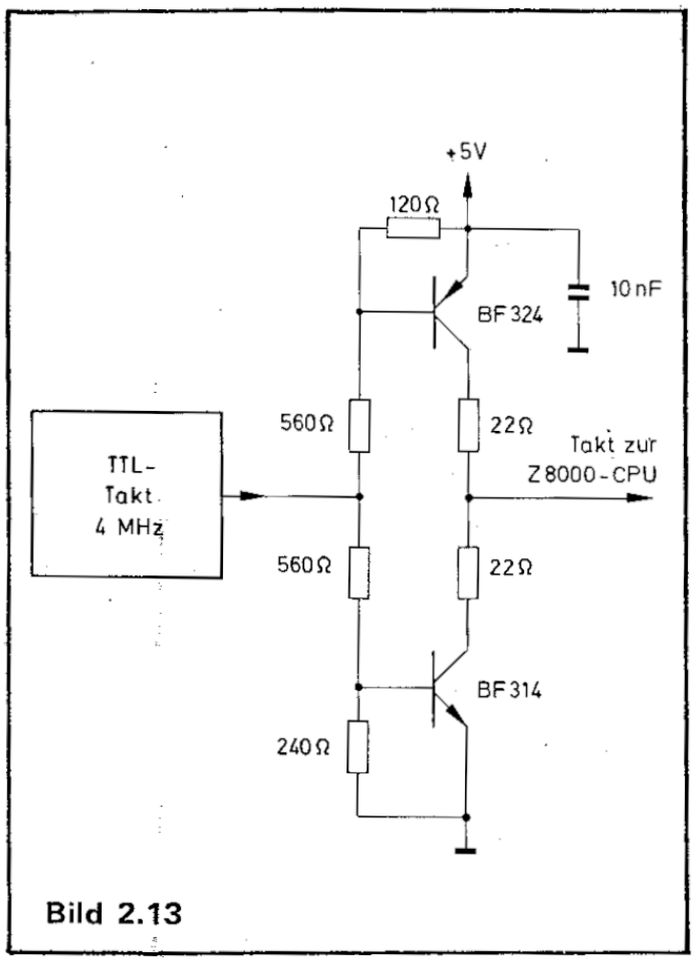


Bild 2.13

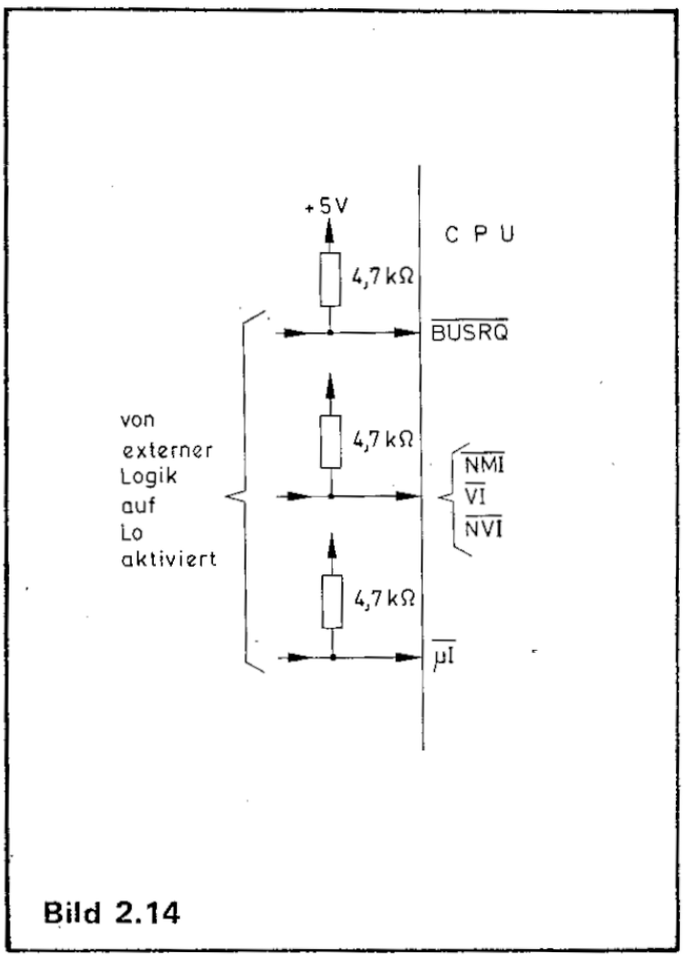


Bild 2.14

bereiteten MOS-Takts. Die beiden komplementären Transistoren haben eine Transitfrequenz von 450 MHz, die für 20 ns Flanken ausreicht.

Decouple (Ausgang):

Die negative Substrat-Bias-Generator-Auskopplung wird nicht weiter verwendet. Eventuelle Schwingneigungen lassen sich mit einem Kondensator von 100 nF gegen Masse ausschalten.

Zusätzliche Anschlüsse an der Z 8001-CPU:

SN 0 . . . SN 6 Segment-Adresse (Ausgänge):

Über diese Anschlüsse wird die Segmentadresse ausgegeben. SN 6 ist das höchstwertige Bit (MSB) und SN 0 das niederwertigste (LSB). Hi entspricht einer Eins und Lo einer Null.

$\overline{\text{SEGT}}$ Segment-Trap (Eingang):

Dieser Anschluß ist für einen Betrieb mit der Z 8010-MMU vorbereitet. Hat die CPU eine komplette Segmentadresse ausgegeben, wird im MMU-Segmentbeschreibungs-Register geprüft, ob diese Adresse einen gültigen Zugriff im Speicher bewirkt. Ist das nicht der Fall, sendet die MMU ein Lo an die CPU über den SEGT-Eingang. In der CPU löst das einen Segment-Trap aus, der an den Statusausgängen einen entsprechenden Code als Anerkennung hervorruft.

Wie man aus der Beschreibung der Anschlüsse bereits entnehmen konnte, sind eine Reihe von Eingängen mit einer wesentlichen Bedeutung versehen, d.h. werden sie aktiviert, vollziehen sich weitreichende Veränderungen an der CPU. Man muß daher vorsehen, daß diese Eingänge einen stabilen, nichtaktiven Zustand einnehmen. Es darf nur die entsprechende Peripherie eine Aktivierung vornehmen. In **Bild 2.14** wird dargestellt welche CPU-Eingänge davon betroffen sind. Die gezeigte Maßnahme einer sicheren Inaktivierung dieser Eingänge ist sehr einfach. Über die 4,7 kΩ-Widerstände werden die Eingänge für die Busanforderung, der Interrupts und des Multi-Mikro-Busses sicher auf Hi gelegt. Man sollte sich diese Maßnahme von vorneherein angewöhnen, um sich beim Betrieb eines selbstgebauten Systems viel Ärger zu ersparen.

2.5 REGISTERAUFBAU UND -FUNKTIONEN

Das Registerfeld einer Zentraleinheit trägt entscheidend zu einer einfachen Entwicklung von Maschinenprogrammen in Assembler bei. Mit den Registern der CPU hat man ständig zu tun. Deshalb ist es nicht verwunderlich, wenn bei der Festlegung von Leistungskriterien zur Beurteilung einer CPU die Registerorganisation eine nicht unbedeutende Rolle spielt. Speziell die Organisation der Allzweckregister (sofern überhaupt Register vorhanden sind, die diese Bezeichnung verdienen) bestimmt die Anzahl der in Abschnitt 2.1 angesprochenen Zubringeroperationen. Zahl und Verwendbarkeit solcher Allzweckregister bestimmen die Übersichtlichkeit eines Programms. Überdies ist die Architektur der Stapelzeiger-Register (Stacks) für den Aufruf von Unterprogrammen ausschlaggebend. Damit wird nur leicht angedeutet, wie wichtig der Abschnitt über Register ist.

Der Z 8000 ist eine Speicher-Register-Maschine. Der Hochleistungs-Zeichenkettenbefehl TRTDRB (übersetzen, testen, wiederholendes dekrementieren eines Quellregisters) ist eine Speicher-zu-Speicher-Operation, dessen Umladung eines Bytes im Speicherfeld über zwei Quellregister (Inhalt Adresse) vorgenommen wird. Es wird dabei **eine** Tabelle von Zeichen mittels eines berechneten Testbytes durchgeprüft. Über ein weiteres Register (benützt als Zähler) wird durch Dekrementieren festgehalten, wie viele Zeichen (Bytes) geprüft wurden. Obschon hier drei Register eine Rolle spielen, sollte im folgenden Beispiel erst von einer besonders typischen Speicher-Register-Speicher-Operation gesprochen werden. Ein anderer Hochleistungs-Zeichenkettenbefehl des Z 8000, der sich CPSDR nennt (Zeichenkettenvergleich zweier Speicherinhalte, wiederholendes Dekrementieren nach Bedingung), vergleicht arithmetisch (absolut) die Werte **zweier** Tabellen. Zwei Register beinhalten die Quell- und Zieladresse eines Wertes in je einer Tabelle, holen sie und er-

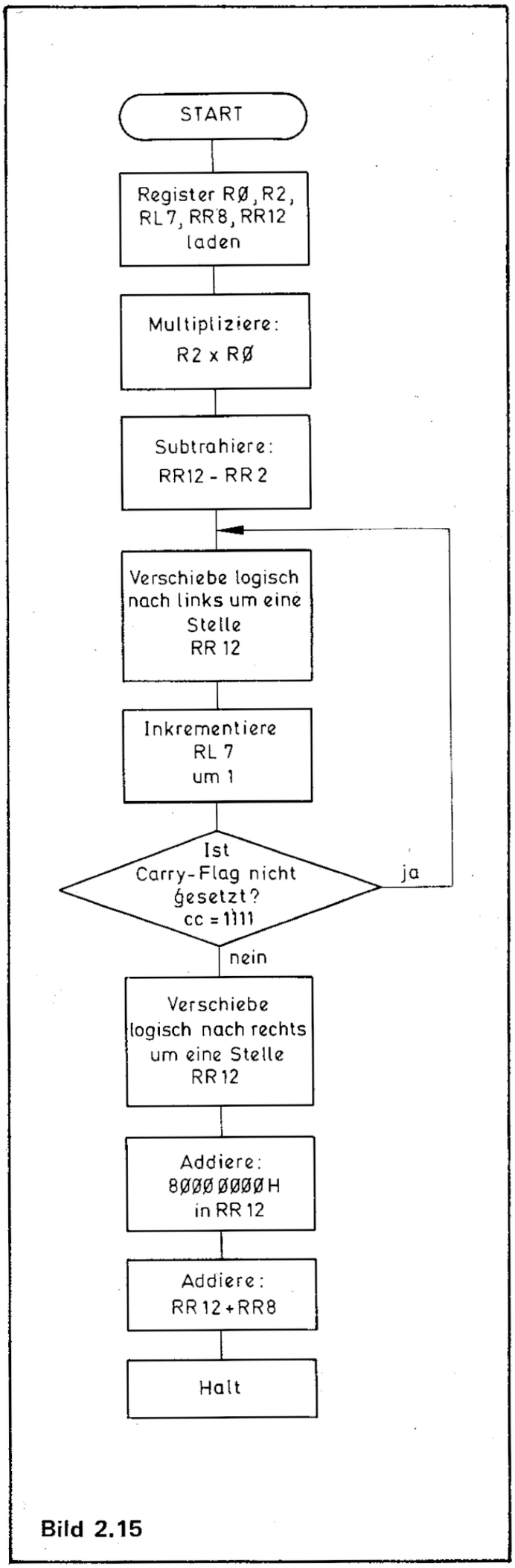


Bild 2.15

mitteln durch eine Subtraktion die Gleichheit. Das Null-Flag (Z-Flag) wird abgefragt. Nach erfolgter Operation werden Quell- und Zielregister jeweils um eins (wenn die Zeichenkette aus Bytes besteht) oder um zwei erniedrigt (wenn die Zeichenkette aus Worten besteht). Ein weiteres Register wird, wie schon bei TRTDRB, als Zähler benutzt, um die Anzahl der verglichenen Zeichen festzuhalten.

Das letzte Beispiel zeigt nun eindeutig, daß solche Befehle nur mit einem hochgradigen Registerfeld möglich sind. Wer hier mitgezählt hat – hier sind fünf 16-bit-Register notwendig.

Um die effektive Bedeutung von Registern aufzuzeigen, wird an einem kleinen Programmbeispiel demonstriert, wie die Z 8000-Register einzusetzen sind. Das Beispiel hat ausgesprochen sporadischen Charakter und sollte deshalb auch nur exemplarisch betrachtet werden. In **Bild 2.15** ist das Flußdiagramm dargestellt. Nachdem zwei Wort-Register, ein Byte-Register und zwei Langwort-Register geladen wurden, wird einmal multipliziert und einmal subtrahiert. Im Ergebnisregister der Subtraktion RR 12 wird dann der Inhalt um eine Stelle nach links verschoben und die Verschiebung durch Inkrementieren eines Byte-Registers registriert. Trat kein Carry (Übertrag) auf, wird durch einen relativen Rücksprung auf den Verschiebebefehl die Operation wiederholt. Trat ein Carry auf, wird RR 12 um eine Stelle nach rechts verschoben und das MSB wieder hinzugezählt. Zuletzt wird RR 12 und RR 8 addiert. Im Byte-Register RL 7 steht die Zahl der Linksverschiebungen. In **Bild 2.16** sind die Operationen grafisch dargestellt. Das Z 8000-Assemblerprogramm sieht für diese Operationen folgendermaßen aus:

Zeile-Nr.	Mnemonic	Bezeichnung	Bemerkung
1	LD	R 0, #%013F	laden
2	LD	R 2, #%100A	
3	LDL	RR 8, #%000020F3	
4	LDB	RL 7, #%00	
5	LDL	RR12, #%11253FB1	
6	MULT	RR 2, R 0	multiplizieren
7	SUBL	RR12, RR 2	subtrahieren
8	SLLL	RR12, #1	schiebe links um 1
9	INC	RL 7, #1	Byte-Reg. + 1
10	JR	No Carry, $ – 8 (SLLL)	PC-8, wenn kein Übertrag
11	SRLL	RR12, #1	schiebe rechts um 1
12	ADDL	RR12, #%80000000	addiere, MSB gesetzt
13	ADDL	RR12, RR 8	addiere
14	HALT		Programm Ende

In Zeile 10 wird der Programmzähler um 8 Stellen zurückgesetzt, damit er auf die Adresse von SLLL zeigt (INC ist ein Wort, SLLL sind zwei Wörter). Das Programm ist sicherlich leicht zu verstehen. Man bemerkt die Beteiligung von Byte-, Wort- und Doppelwort-Registern. Würde man dieses Programm mit einem 8080 lösen wollen, wäre der Programmierer vor eine "echte Aufgabe" gestellt. Dieses Programm läuft auf einem 8001- oder 8002-System gleichermaßen. Es zeigt sich hier, daß keine unnötigen Vorbereitungsbefehle im Programm stehen.

Wenn es auch nur ein kleines Beispiel ist, zeigt es doch die enorme Leistungsfähigkeit der Z 8000-CPU. Die Register können universell verwendet werden.

Das gesamte Registerwerk des Z 8000 ist in Allzweck-Register, Normal-/System-Stapelzeiger, Programmstatus-Register, Programmstatus-Bereichszeiger und einem Refresh-Steuer/Zähler-Register zu unterscheiden. Auf Grund der Segmentadressierung des Z 8001 erge-

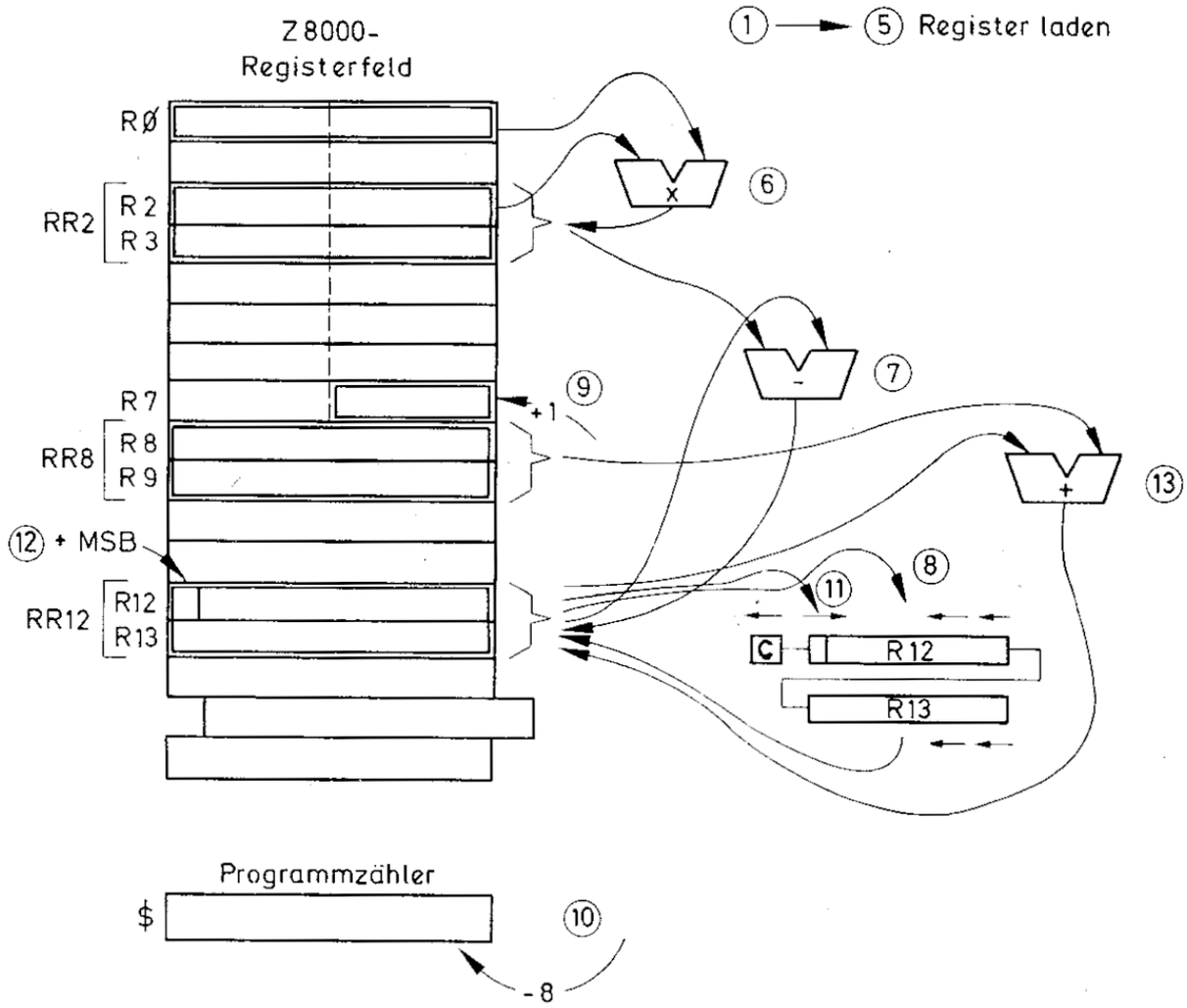


Bild 2.16

ben sich gewisse Unterschiede im Registerwerk beider Versionen. Die Unterschiede liegen in den Status-Registern, von denen der Z 8001 vier und der Z 8002 zwei 16-bit-Register besitzt. Bei den Status-Bereichszeigern hat der 8001 zwei und der 8002 ein 16-bit-Register. Unterschiede ergeben sich auch in den Stapelzeigern beider Versionen. **Bild 2.17** zeigt den Allzweck-Registersatz des Z 8001 mit seinen Stapelzeigern für Segment und Offset (jeweils für Normal- und Systemmode). **Bild 2.18** zeigt den Allzweck-Registersatz des Z 8002, der nur je einen Normal-/System-Stapelzeiger besitzt. Grundlegend ist jedoch bei beiden Versionen, daß sowohl Byte-, Wort-, Doppelwort als auch maximal 4 Vierfachwort-Register (64 bit Länge) zur Verfügung stehen. Letztere stehen nur als Ergebnisregister für die Langwort-Multiplikation und -Division zur Verfügung. Sie werden mit Wort-Registern in einer Serienschaltung gebildet. Bild 2.17

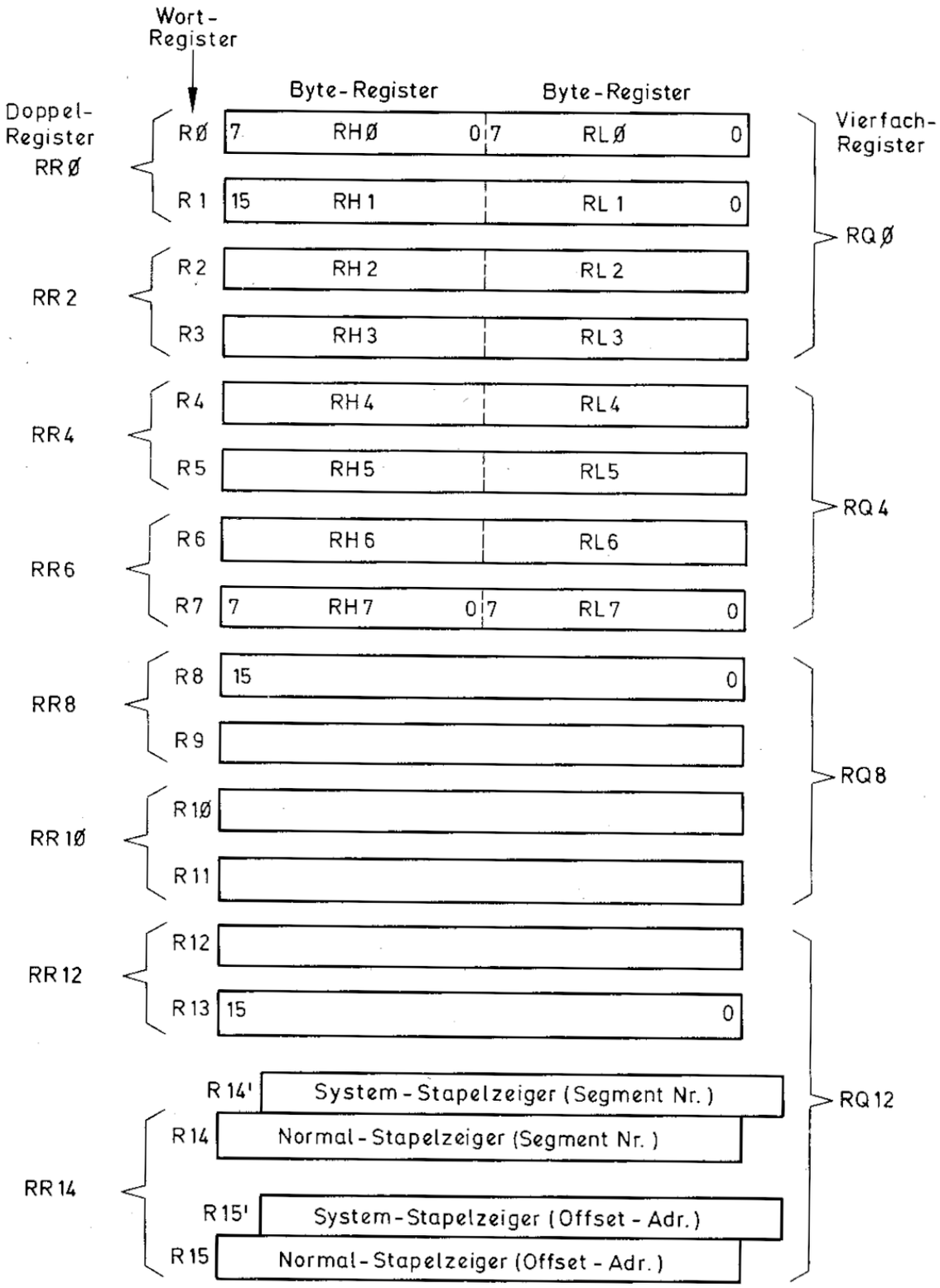

Wort-
Register
Byte-Register
Byte-Register
Doppel-
Register
Vierfach-
Register
RRØ
RØ
7
RHØ
0
7
RLØ
0
R1
15
RH1
RL1
0
RQØ
RR2
R2
RH2
RL2
R3
RH3
RL3
RR4
R4
RH4
RL4
R5
RH5
RL5
RQ4
RR6
R6
RH6
RL6
R7
7
RH7
0
7
RL7
0
RR8
R8
15
0
R9
RQ8
RR1Ø
R1Ø
R11
RR12
R12
R13
15
0
R14'
System-Stapelzeiger (Segment Nr.)
RQ12
RR14
R14
Normal-Stapelzeiger (Segment Nr.)
R15'
System-Stapelzeiger (Offset-Adr.)
R15
Normal-Stapelzeiger (Offset-Adr.)

Bild 2.17

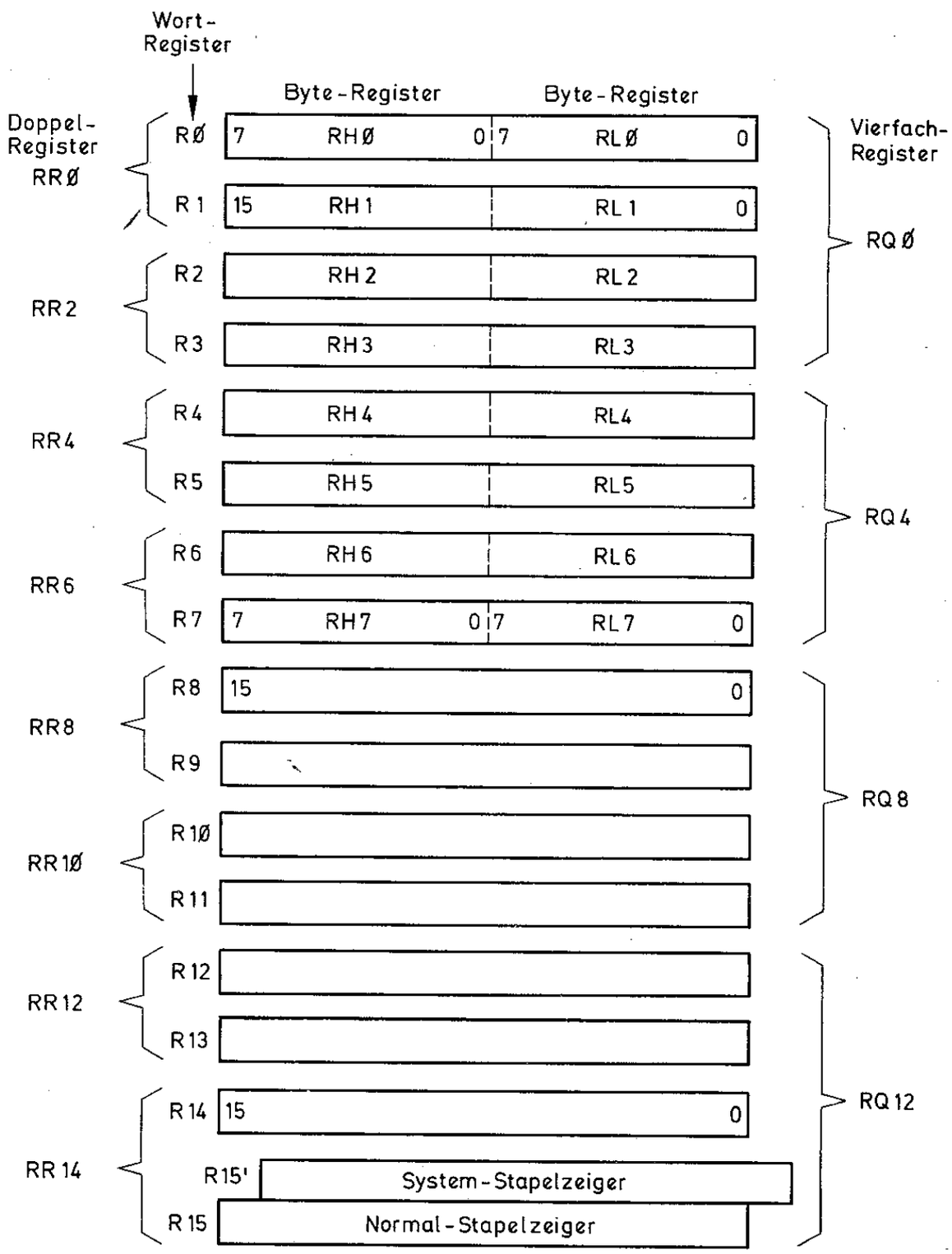

Wort-
Register
Byte-Register
Byte-Register
Doppel-
Register
Vierfach-
Register
RØ
RRØ
7
RHØ
0
7
RLØ
0
R1
15
RH1
RL1
0
RQØ
R2
RH2
RL2
RR2
R3
RH3
RL3
R4
RH4
RL4
RR4
R5
RH5
RL5
RQ4
R6
RH6
RL6
RR6
R7
7
RH7
0
7
RL7
0
R8
15
0
RR8
R9
RQ8
RRIØ
RIØ
R11
R12
RR12
R13
RQ12
R14
15
0
RR14
R15'
System-Stapelzeiger
R15
Normal-Stapelzeiger

Bild 2.18

und 2.18 zeigen, welche Wort-Register zusammengeschalten werden. Doppelwort-Register (32 bit Länge) stehen hingegen für eine Vielzahl von Operationen zur Verfügung. Doppel- oder Langwortoperationen bilden einen breiten Bestandteil des Z 8000-Befehlssatzes. Andererseits stehen insgesamt 16 Byte-Register zur Verfügung, die jeweils durch Halbieren der ersten acht Wort-Register gebildet werden. Diese Byte-Register ergeben einen eigenständigen Registersatz. Ein überwiegender Teil der Z 8000-Befehle sind auch im Bytemode programmierbar. Im Bytemode sind überdies ausgesprochene exotische Befehle anwendbar. DAB und RRDB bzw. RLDB ermöglichen Operationen mit Digits. Die CPU bearbeitet in diesem Fall 4 bit BCD-Digits in Byte-Registern. Die beiden Rotier-Befehle lassen beispielsweise sehr leicht Multiplikationen bzw. Divisionen mit dem Faktor 10 für BCD-Werte zu.

Den Ausgangspunkt aller Registervariationen bilden die 14 bzw. 15 Wortregister. Man kann sie mit Operandenadressen oder Daten laden. Mit Ausnahme des Registers R 0 ist jedes als Index-Register verwendbar. Ab Register R 2 ist jedes als Anwender-Stapelzeiger für PUSH- und POP-Operationen einzusetzen, wie **Bild 2.19** zeigt. Für die Stapelzeiger gilt als genereller Einsatz der Aufruf oder ein Rücksprung einer bzw. aus einer Service-Routine, UPRO oder wie man es sonst nennen mag. CALL und RET sind die maßgebenden Befehle im Zusammenhang mit den Stacks.

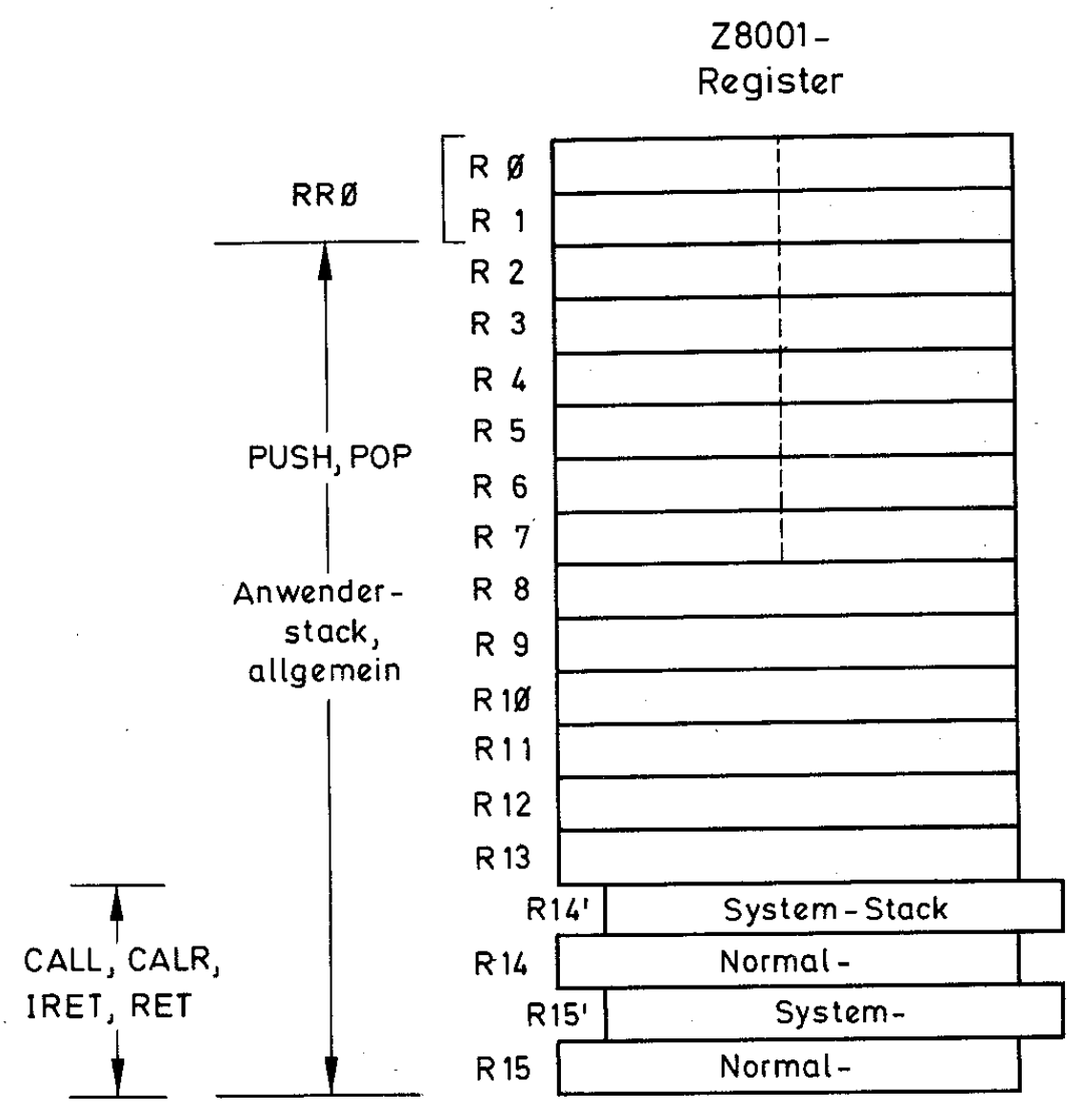


Bild 2.19

Die Adressierungsarten indexed (relative oder indizierte Adressierung), Basis-Adressierung oder die Basis indizierte Adressierung (speziell bei Ladebefehle angewandt) sind für die **Register R 0** bzw. **RR 0 nicht** anwendbar. Dies ist eine Regel, die man beim Programmieren des Z 8000 auf Maschinenebene beachten muß. Wie die genannten Adressierungsarten zu unterscheiden sind, wird in Kapitel 3 dieses Buches behandelt. Die Bezeichnung Allzweckregister für R 0 . . . R 15 gilt trotz der kleinen Einschränkung als absolut nachdrücklich. Die Kurzformel Akku, Index, Stack läßt spätestens jetzt erkennen, wie vielfältig die Verbindungen des Registerwerks mit anderen Logikfunktionsgruppen auf dem Z 8000-Chip sein müssen.

Ein quantitativ kleineres Feld des Registerwerks nehmen die Programmstatusregister ein. In dem vorangegangenen Beispiel wurden sie ohne weitere Erklärung benutzt und zuvor mehrfach erwähnt. Die Programmstatusregister bestehen aus zwei Elementen, dem Programmzähler (PC) und dem Flag- und Steuerwort-Register (FCW). Diese 16-bit-Register haben eine so elementare Bedeutung, daß kein Programm ohne sie läuft. Die Verknüpfung mit den Allzweckregistern ist für jeden einzelnen Bit für Bit vorhanden. Ohne PC gibt es kein stetig ablaufendes Programm. Ohne FCW ist jegliche Definition einer Operation unterbunden. Kurzum, ohne PC und FCW geht gar nichts.

Auch für die Programmstatusregister gilt es, zwischen der segmentierten und der unsegmentierten CPU-Version zu unterscheiden. Beim Z 8001 besteht der primäre Status aus drei Registern (ein FCW und 2 PC's). Beim Z 8002 sind nur zwei Register notwendig (ein FCW und ein PC). Der Unterschied zum Z 8001 besteht in je einem Register für die Offsetadresse und den Segmentadreßteil. In Bild 2.10 ist der Z 8001-Programmzähler in seiner Art bereits dargestellt worden. Bliebe an dieser Stelle nur noch zu bemerken, daß der Programmzähler selbstverständlich individuell gesetzt werden kann. Dies geschieht mit dem priviligierten Ladebefehl LDPS (lade Programmstatus). Hierbei wird zwischen der segmentierten und nichtsegmentierten Betriebsart unterschieden, nicht pauschal zwischen beiden CPU-Versionen. Der Befehl LDPS bewirkt das direkte Laden des FCW und des PC.

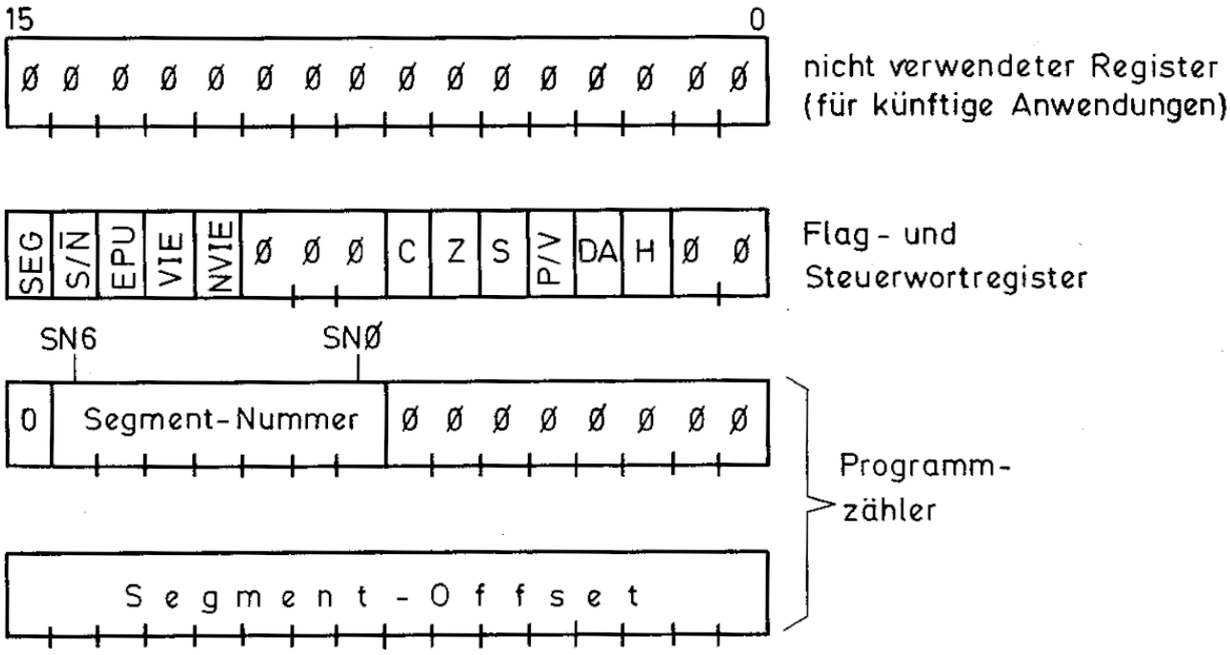
Das Flag- und Steuerwortregister FCW ist in zwei Bytes zu trennen, wie in der Bezeichnung bereits angedeutet wird. Das höherwertige Byte beinhaltet zu definierende System-Statusbits, die die Betriebsart der CPU festlegen. Das niederwertige Byte bezeichnet Programm-Steuerflags, die die Ausführung eines Programms unmittelbar beeinflussen. Sie werden bei der Ausführung von Befehlen über die ALU (arithmetische Logikeinheit) gesetzt oder direkt über einen Ladebefehl initialisiert. Damit ist eine Rangordnung beider Bytes schon angedeutet. Sie ist gleichbedeutend mit der Ordnung der Bytes im FCW. Die System-Statusbits werden über einen priviligierten Ladebefehl einmal eingegeben und bleiben bis zur nächsten Initialisierung unverändert, wogegen die Flags auch vom Programm selbst geändert werden. Über den Befehl LDCTL (lade Steuerwort) geschieht die Initialisierung individuell. Mit einem Hardware-RESET initialisiert sich die CPU selbst.

In der folgenden Auflistung werden die einzelnen Bits des FCW vorgestellt, – zuerst die Flags und daran anschließend die System-Statusbits. In **Bild 2.20** und **2.21** sind die Programmstatus-Register, die Status-Bereichszeiger und das Refresh-Register gemeinsam für die jeweilige CPU dargestellt.

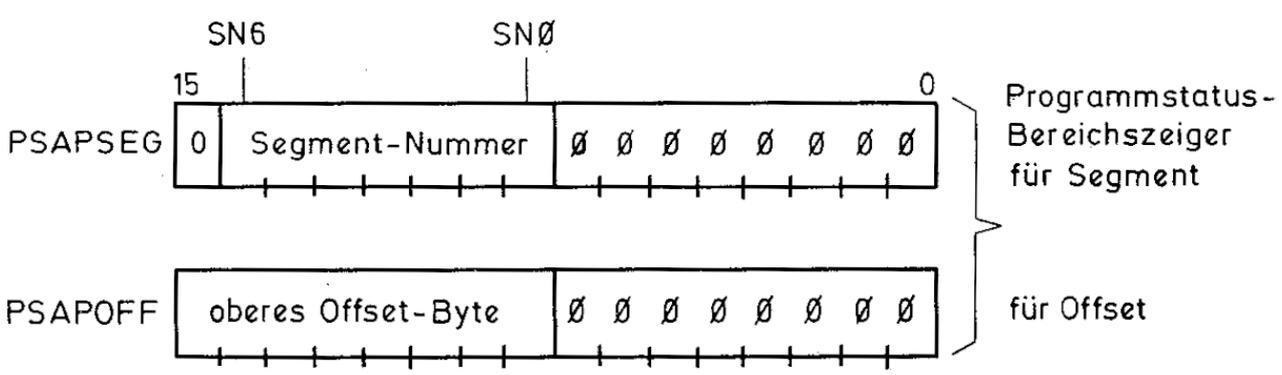
Statusregister-Flags:

C (Bit 7):
Carry gleich 1 zeigt an, daß das höchstwertige Bit (MSB) bei einer arithmetischen Operation (addieren, schieben, rotieren) als Übertrag gesetzt wurde.

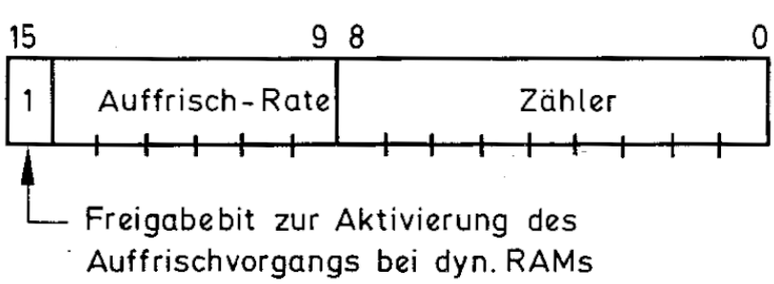
Z (Bit 6):
Zero gleich 1 zeigt an, daß bei einer Vergleichsoperation (Subtraktion) oder bei



Z8001 Programm-Status-Register

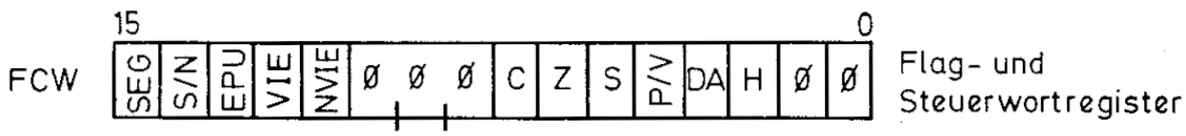


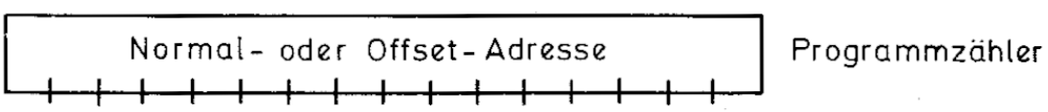
Z8001 Programm-Status-Bereichszeiger



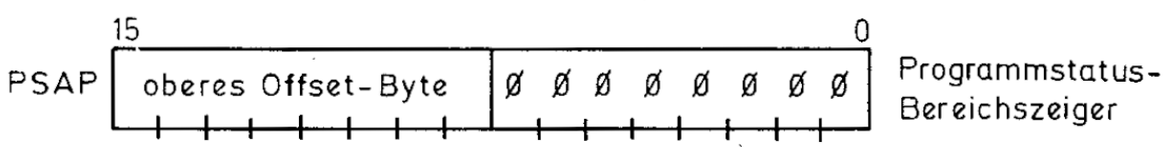
Z8001 Refresh-Register

Bild 2.20

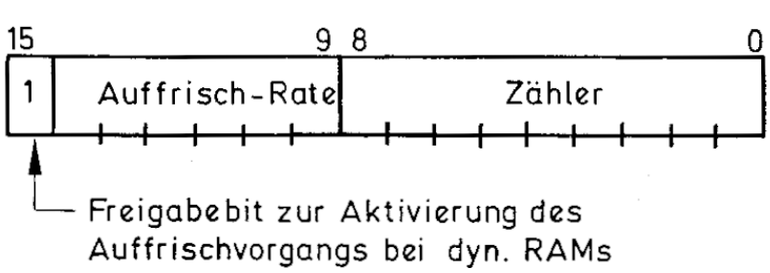




Z8002 Programm-Status-Register



Z8002 Bereichszeiger



Z8002 Refresh-Register

Bild 2.21

wiederholtem Dekrementieren als Ergebnis Null entstanden ist, bzw. Null erreicht wurde.

S (Bit 5):
Vorzeichen gleich 1 zeigt an, daß z.B. bei einer Addition, Subtraktion oder eines Vergleichs (CP) ein negatives Ergebnis entstand oder bei dem Vergleich "kleiner als . . ." heraus kam.

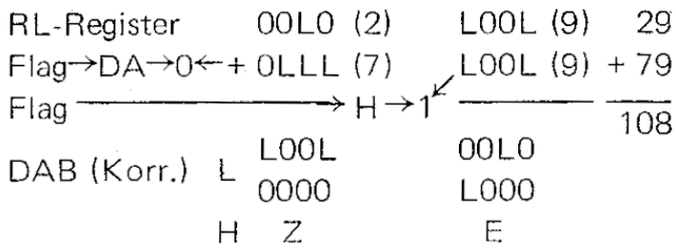
P/V (Bit 4):
Parität/Überlauf gleich 1 zeigt an, daß während einer arithmetischen Operation ein Überlauf entstanden ist. Ein Überlauf entsteht, wenn beispielsweise zwei Operanden gleichen Vorzeichens addiert werden und im Ergebnis ein anderes Vorzeichen entstanden ist, was ja bekanntlich falsch wäre. Für logische Byte-Operationen zeigt dieses Flag eine Parität an; z.B. bei ANDB wird es gesetzt, wenn das Ergebnis gerade ist.

Für diese vier bedeutenden Flags C, Z, S und P/V sind drei CPU-Steuerbefehle im Befehlssatz inbegriffen, die sich nur auf diese Flags beziehen.
Der Befehl COMFLG bildet das Komplement jedes aufgeführten Flags (0 wird zu 1 und umgekehrt. Der Befehl RESFLG setzt alle aufgeführten Flags auf Null. Der Befehl SETFLG schließlich erlaubt das Setzen aller aufgeführter Flags auf den Wert 1. Beim Z 80 gibt es vergleichsweise hierzu nur die Befehle CCF (Komplement Carry) und SCF (Carry setzen). Der Status verschiedener Flags ist bei einzelnen Prozeduren sehr wichtig. Es genügt nicht nur zu fragen, wie die einzelnen Flags nach einer Operation stehen. Oftmals muß ein bestimmtes Flag vor einer Operation genau definiert sein, damit nach der Operation sein Status richtig interpretiert werden kann. Bei dem Programmbeispiel in diesem Abschnitt ist der Status des Carry nur nach Ausführung der Operation von Bedeutung.

Die Befehle zur Variation der Flags des Z 8000 erlauben eine einfache Handhabung.

DA (Bit 3):
Dieses Flag wird aktiviert, wenn eine Byte-Arithmetik durchgeführt wird. Bei dem Befehl ADDB (addiere Bytes) wird DA auf Null gesetzt. Bei dem Befehl SUBB (subtrahiere Byte) wird DA auf eins gesetzt. DA wird im Zusammenhang mit dem Befehl DAB (dezimale Korrektur einer Binärarithmetik) verwendet, wenn die Bits eines Byteregisters als eine Binärverknüpfung zweier BCD-Digits verstanden werden soll.

H (Bit 2):
Dieses Flag bezeichnet ein Carry einer BCD-Arithmetik (Addition und Subtraktion). Es wird als Halb-Carry-Flag bezeichnet und liefert einen Übertrag von Bit 3. Es gilt nur für die Byte-Arithmetik. Folgendes zeigt den Zusammenhang:



Nachdem die bei Digits addiert wurden, ergibt sich das übliche binäre Ergebnis. Die Addition selbst setzt das DA-Flag auf Null und das H-Flag auf 1, weil an der vierten Stelle ein Übertrag auftrat. Der Befehl DAB korrigiert nun die beiden Digits auf die richtigen BCD-Werte.

Die Z 80-CPU hat mit N und H gleichbedeutende Flags zu den letztgenannten des Z 8000. Ein Befehl des Typs DAB fehlt jedoch. Er ist auch bei anderen Prozessoren nicht generell zu finden. Über den nicht privilierten Befehl LDCTLB (lade Flag-Byte) kann der Flag-Abschnitt des FCW separat initialisiert werden. Entweder wird er von einem Byteregister geladen oder als momentaner Zustand in ein Byteregister abgespeichert.

In der folgenden Auflistung werden die System-Statusbits des FCW beschrieben:

SEG (Bit 15):
Dieses Bit wurde in Abschnitt 2.3 bereits erwähnt. Es gilt ausschließlich für den Z 8001. Wird dieses Bit auf 1 gesetzt, arbeitet die CPU im segmentierten Modus. Ist es Null, arbeitet die CPU im nicht-segmentierten Modus. Im FCW-Register des Z 8002 ist zwar ebenfalls die Bezeichnung SEG enthalten; sie wird jedoch immer Null gesetzt.

S/$\overline{N}$ (BIT 14):
Wenn man von einer Art Rangordnung der System-Statusbits sprechen will, ist es schwer definierbar, ob SEG oder S/$\overline{N}$ mehr Bedeutung hat. Dies ist auch unerheblich, denn beide Betriebsarten sind völlig voneinander unabhängig. Die 110 Befehlstypen mit ihren insgesamt 414 Varianten enthalten eine Reihe priviligierter Befehle. Dieser "Privileg-Block" ist **nur** in der **System-Betriebsart** ausführbar. Dazu muß S/$\overline{N}$ gleich 1 gesetzt werden. Dies gilt für beide CPU-Versionen. Eine weitere Erklärung beider Betriebsarten System und Normal (S/$\overline{N}$ gleich 0) wird in Abschnitt 2.7 über die Betriebszustandsanzeige gegeben.

EPU (Bit 13):
Ist dieses Bit auf 1 gesetzt, wird ein erweiterter Befehlssatz wirksam (6 Befehle), der für die EPU gilt (**E**xtended **P**rocessing **U**nit bedeutet Entlastungs- oder Parallelprozessor zur CPU). Dieser Status zeigt global an, daß das Computersystem aus einer CPU und einem (oder mehreren) Entlastungsprozessoren besteht. Ist dieses Bit Null, arbeitet die CPU im "Normal"-Betrieb (S oder N) und zugleich über einen Extended Instruction Trap als Software-Emulator (softwaremäßige Nachbildung) einer EPU.

VIE (Bit 12):
Ist dieses Bit auf 1 gesetzt, akzeptiert die CPU eine vektorisierte Interrupt-Anforderung eines peripheren Systems.

NVIE (Bit 11):
Ist dieses Bit auf 1 gesetzt, akzeptiert die CPU eine nicht-vektorisierte Interrupt-Anforderung eines peripheren Systems.

Die Bits 8, 9, 10 sowie 0 und 1 des FCW-Registers sind bis zur Fertigstellung dieses Buches als nicht belegt angegeben worden. Es ist jedoch möglich, daß sie künftig zur Freigabe für (noch) nicht implementierter Befehle verwendet werden.

Nach den Programm-Statusregistern folgt die Beschreibung der Programmstatus-Bereichszeiger, wie sie in Bild 2.20 und 2.21 für die jeweilige CPU-Version gezeigt sind. Die Unterscheidung erfolgt wiederum nach der Segmentierung oder Nicht-Segmentierung. Die Programmstatus-Bereichszeiger-Register (PSAP) haben die Aufgabe, nach einem Trap, einem UPRO-Aufruf und insbesondere nach einer Interrupt-Anforderung zu einem bestimmten Wort in den Statusspeicherbereich zu zeigen.

An dieser Stelle sollte ein wenig zu dem Unterschied zwischen Traps und Interrupts erklärt werden und darüberhinaus, wie der Z 8000 Unterbrechungen eines Programms behandelt.

Es gibt also zwei Arten von Unterbrechungen eines laufenden Programms (Traps und Interrupts). Esskann erforderlich sein, in einem anderen Programm weiter zu arbeiten, weil eine bestimmte Konstellation der laufenden Ergebnisse einen Übertritt in das andere erfordert. Dieser Sprungpunkt kann vorhersehbar sein. Es ist eine in Grenzen vorhersehbare Unterbrechung des laufenden Programms. Sieht ein Computer solch ein Ereignis herannahen, wird diese Art Unterbrechung der laufenden Arbeit als Trap bezeichnet. Traps sind synchrone Ereignisse und folgen entweder aus einer bestimmten Bedingung oder eines Programmierfehlers. Kennzeichnend ist überdies, daß der Sprung nicht programmiert war. Er wird durch die hardwaremäßige Dekodierung automatisch an einer vorbestimmte Stelle ausgelöst. Ein Trap ist jedoch nicht immer die Reaktion eines Fehlers im Pro-

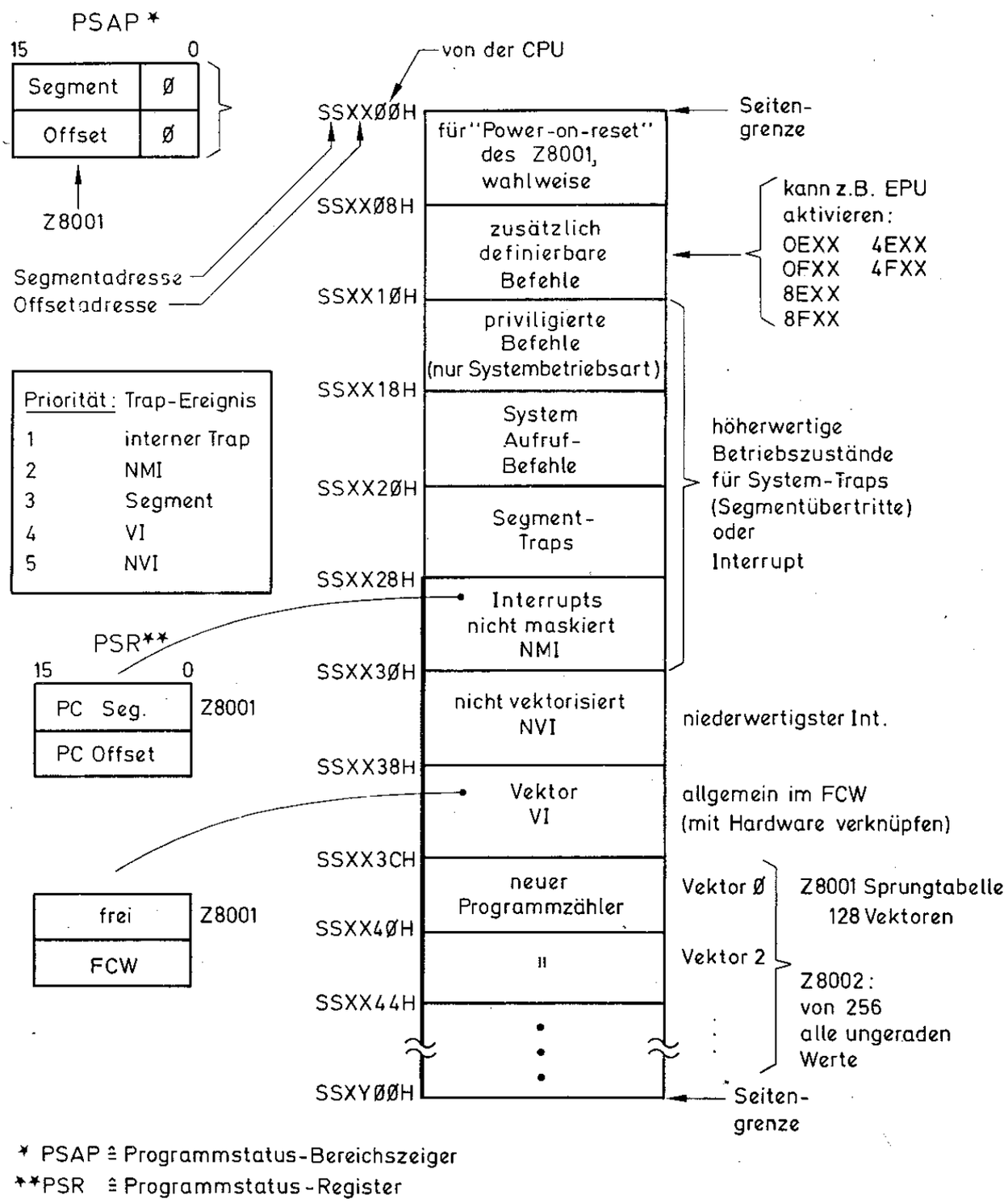
gramm. Man kann eine Reihe von Fehlern im Programm begehen, ohne daß ein Trap ausgelöst wird. Die CPU verfängt sich möglicherweise irgendwosin einer Schleife, so daß sie nur über einen RESET aus dieser gerettet werden kann.

Ein Interrupt im allgemein üblichen Sprachgebrauch der Mikrocomputertechnik ist in der Tat etwas anderes. Ein Interrupt ist ein absolut (für den Computer) unvorhergesehenes Ereignis, das den regulären Programmablauf eines Computers anhalten kann. Ein Interrupt ist ein asynchrones Ereignis, das ausschließlich von einem peripheren System initiiert wird. Je nach der Priorität des Interrupt muß die CPU darauf reagieren. Wie reagiert nun die CPU auf dieses Ereignis?

Jedes Ereignis fordert normalerweise eine Reaktion des Betroffenen heraus. Eine völlig menschlische und jedermann einleuchtende Reaktion ist, wenn man bei einer Aufgabe unterbrochen wird, eine Notiz zu machen, um bei einem späteren Weiterarbeiten wieder leichter den Anknüpfungspunkt zu finden. Diese Denkweise wurde in die Architektur eines Prozessors dadurch übernommen, indem man den Stapelzeiger einführte. Was dieser bewirkt, ist prinzipiell nichts anderes, als beispielsweise einen kleinen Papierstreifen in ein Buch zu legen, um später die Fortsetzung schnell wieder finden zu können. Über den Stapelzeiger werden der Segment- und Offset-Adreßteil des letzten PC-Standes und der entsprechende FCW-Zustand gerettet. Man kann das noch weiter ausdehnen, indem man die Registerinhalte mit dem Befehl LDM in den Speicher hinauspusht. Diese Methoden sind bekannt.

Beim Z 8000 ging man noch eine Stufe höher. Nach der Unterbrechung eines Programms (Trap oder Interrupt) muß man auch den neuen Programmstatus (mindestens Segment- und Offset-Adreßteil) laden, um die Aufgabe zu bearbeiten, die die Unterbrechung erfordert. Wenn man von Ausnahmen absieht, hat im allgemeinen jedes Ereignis einen Grund. So auch hier. Es wird über den PSAP die Startadresse geladen, die den Grund einer bestimmten Art von Unterbrechung mit enthält. Der Z 8000 ist in der Lage, zwischen Trap und Interrupt sehr wohl zu unterscheiden. Nicht nur dies. Er unterscheidet zwischen verschiedenen Traps und verschiedenen Interrupts. Seine Reaktion darauf ist die Ausgabe einer ganz bestimmten Adresse für eine Service-Routine auf Grund eines ganz bestimmten Ereignisses. Man unterscheidet zwischen Gründen, die einen Trap hervorrufen und denen, die einen Interrupt auslösen. Der Z 8000 macht darin einen feinen Unterschied. Der PSAP ist ein Zeigeregister, das auf einen bestimmten Programmstatusbereich weist, der das anzusteuernde Ziel einer Unterbrechung darstellt. In dieser Programmstatus-Tabelle stehen Befehle, die das Weiterarbeiten der CPU im Fall eines Trap garantieren oder eine Interrupt-Serviceroutine enthalten, deren Bedarf eine Interrupt-Anforderung anmeldet. Der PSAP-Zeiger besteht aus zwei bzw. einem Wort (Segment, Offset), wobei jeweils Nullen im unteren Byte des Zeigerregisters stehen. Dieses untere Byte wird von der CPU ausgegeben (im Fall eines Trap), oder es wird von der entsprechenden Peripherie auf den Datenbus gegeben, die einen vektorisierten Interrupt ausgelöst hat. Dies ergibt dann ein bestimmtes Zeigerwort (Adresse), das die Trap- oder Interrupt-Serviceroutine startet. Die Ergänzung von Bit 0 . . . Bit 7 wird von der CPU selbst oder von der Peripherie gegeben, genauer gesagt, der Befehl (erstes Wort), der den Trap verursachte, liefert der Zeiger innerhalb des 256 Byte-Blocks. Bei einem Interrupt liefert dieses Byte den Inhalt auf den AD-Leitungen in einem Teilzyklus der Interrupt-Anerkennungs-Prozedur. Der benötigte Speicherblock für den Programm-Status-Bereich kann vom Programmierer frei definiert werden.

Wie dieser Speicherblock prinzipiell organisiert ist, zeigt **Bild 2.22.** Von der Seitengrenze weg ist dieser Block 256 Worte lang. Die Darstellung der Pro-



Programmstatusbereich: Sprungtabelle für den Z8001.
Die Werte SSXX werden in den Programmstatus-Bereichszeiger PSAP geladen.
Der unterste 8 bit Wert wird von der Z8001 CPU hinzuaddiert.
Für den Z8002 halbieren sich die Werte.

Bild 2.22

grammstatusbereiche bezieht sich auf die segmentierte Form. Die von SSXX00H . . .07H ist eigentlich freibleibend. Wie aber hier angegeben, kann in diesem Bereich eine Routine für den Power-on-reset angesiedelt werden. Von . . . 09H . . . 0FH sind nicht implementierte Befehle einzubauen. Hier sind als Beispiel 6 Befehle für die EPU vorzusehen. Weiter oben im Feld . . . 3CH beginnt dann die Sprungtabelle für den PC auf Grund einer Vektor-Interrupt-Anforderung.

Die Tabelle von Adressen bilden den Block des Programmstatusbereichs-Speichers, der von dem Programmstatus-Bereichszeiger (PSAP) adressiert wird. Wenn also ein Trap oder Interrupt aufgetreten ist, werden erst der momentane Programmstatus in den System-Stack gerettet und dann der neue Programmstatus der entsprechenden Serviceroutine automatisch in den PC geladen. Dies wird über den PSAP gemacht. Dazu wird der Stack-Zeiger nicht mißbraucht. Nachdem die Serviceroutine bearbeitet wurde, wird in das vorhergehende Programm über IRET oder RET zurückgekehrt, indem der alte Programmstatus (noch im System-Stack stehend) wieder in den PC geladen wird. Das vor dem Trap bzw. Interrupt bearbeitete Programm kann darauf hin wieder weiter bearbeitet werden.

Diese Architektur von Stacks und Programmstatus-Bereichszeiger ist bei anderen Mikroprozessoren nicht zu finden. Bei einem entsprechenden Unterbrechungsereignis kann der alte PC-Stand im System-Stack-Speicher oben an stehenbleiben, weil der neue Programmstatus über den PSAP geholt wird. Schneller gehts nicht. Eine schnelle und sichere Rückkehr ist ebenfalls gesichert.

Nach der vorangegangenen Beschreibung könnte es dem Leser womöglich den Eindruck vermitteln, nur mit dem Z 8001 wären die genannten Operationen möglich. Das ist natürlich falsch. Es besteht nur die eine Bedingung, daß eine Interrupt-Anforderung nur im System-Modus ausgeführt wird. Beim Z 8001 kommt noch die segmentierte Betriebsart hinzu, unbeachtet des vorhergehenden Statusses. Der Rettvorgang nach Anerkennung des Interrupt oder Trap ist nur geringfügig unterschiedlich. **Bild 2.23** zeigt den Unterschied. Enthält der Systemstapelzeiger, wird zuerst PC-Offset, dann PC-Segment, weiter der FCW-Inhalt und zuletzt die Kennung gerettet. Somit steht der System-Stack auf $ - 8, weil es insgesamt 8 Bytes sind. Das Kennungswort ist entweder das erste Wort eines Befehls, das den Trap verursachte oder ein Initialisierungswort des Interrupts. Für einen NMI oder NVI kann es ein Kennungswort in diesem Sinn sein. Es kann auch ein Datenwort sein. Für einen VI ist das niederwertige Byte der Sprungvektor. Wird nach Beendigung der Unterbrechungsroutine zurückgekehrt, steht der Systemstack immer noch auf $ - 8 (beim Z 8002 auf $ - 6). Er zeigt noch immer auf die Kennung. Beim Laden des alten Programmstatus wird in der umgekehrten Reihenfolge verfahren, wie zuvor bei der INT/Trap-Anerkennung. Die "zuletzt rein, zuerst wieder hinaus"-Arbeitsweise (LIFO) besitzen auch die Z 8000-Stacks.

Bei einem Prozessor wie den Z 8000 muß es geradezu selbstverständlich erscheinen, für den Betrieb dynamischer RAMs gewisse Hilfen zu bieten. Dynamische RAMs erfordern bekanntlich das stetig pulsierende Anlegen einer Adresse, die die kapazitiven Speicherzellen immer wieder anregt. Unterbleibt dies, ist der Inhalt der Zellen mit dem Wert 1 unwiederbringlich verloren. Der Z 8000 mit seinem extrem großen, adressierbaren Speicherfeld erfordert den Einsatz von dynamischen RAMs. Das Speicherfeld mit statischen Schreib-/Lese-Speichern auszufüllen, wäre zu kostenintensiv. Schon die Z 80-CPU weist für den Betrieb dynamischer RAMs Hilfsmittel auf. Ein 7-bit-Zählerregister wird nach jedem Holzyklus automatisch inkrementiert.

Während der Befehlsdekodierung wird dieser 7-bit-Wert als Adresse am unteren Byte des Adreßbusses ausgegeben und am Ausgang $\overline{RFSH}$ signalisiert. Diese 7-bit-Adresse kann als Zeilenrefresh für 16 K x 1 dynamische RAMs verwendet werden. Wie macht das der Z 8000? Sieht man sich Bild 2.7 nochmal an, kann zumindest an seinen äußeren Anschlüssen bezüglich Refresh nichts festgestellt werden. Der Z 8000 hat das auch nicht nötig. Der Z 8000 enthält ein 16-bit-Auffrisch-Register, das für einen automatischen Refreshzyklus alles enthält, was notwendig ist. Die Refresh-Adresse wird auf der Breite des gesamten AD-Busses ausgegeben. Wie Bild 2.20 und 2.21 zeigt, ist dieses Register für beide CPU-Versionen identisch. Es ist in die Abschnitte Freigabe (1 Bit), Auffrischrate (6 Bit) und Zähler (9 Bit) unterteilt. Der 9-bit-Zeilenzähler (RC) kann 256 Zeilen adressieren, d.h. er ist bereits für 64 K x 1 dynamische RAMs ausgelegt. Er wird immer um 2 inkrementiert, wenn der 6-bit-Ratezähler seinen Endwert erreicht hat. Der Ratezähler definiert die Zeit, nach der jeweils ein Auffrischzyklus erfolgt ist. Er wird mit dem vierten Teil der Taktfrequenz der CPU dekrementiert und liefert 64 verschiedene Auffrischraten, die von 1 . . . 64 μs in Schritten einer Mikrosekunde reichen. In **Bild 2.24** ist dargestellt, wie die Refresh-Zyklen gebildet werden. Der 2^0-Wert des Zeilenzählers RC ist immer Null, d.h. bei der Refresh-Adreßausgabe wird demnach immer eine gerade Adresse erscheinen, so daß immer nur das Zeilenadreß-Register in den dynamischen RAMs geladen wird. Voraussetzung ist dabei eine entsprechende Zeilenadreß-Multiplexschaltung, auf die noch eingegangen wird. Ferner muß das Auffrisch-Freigabebit Nr. 15 gesetzt sein, sonst erfolgt kein Refresh.

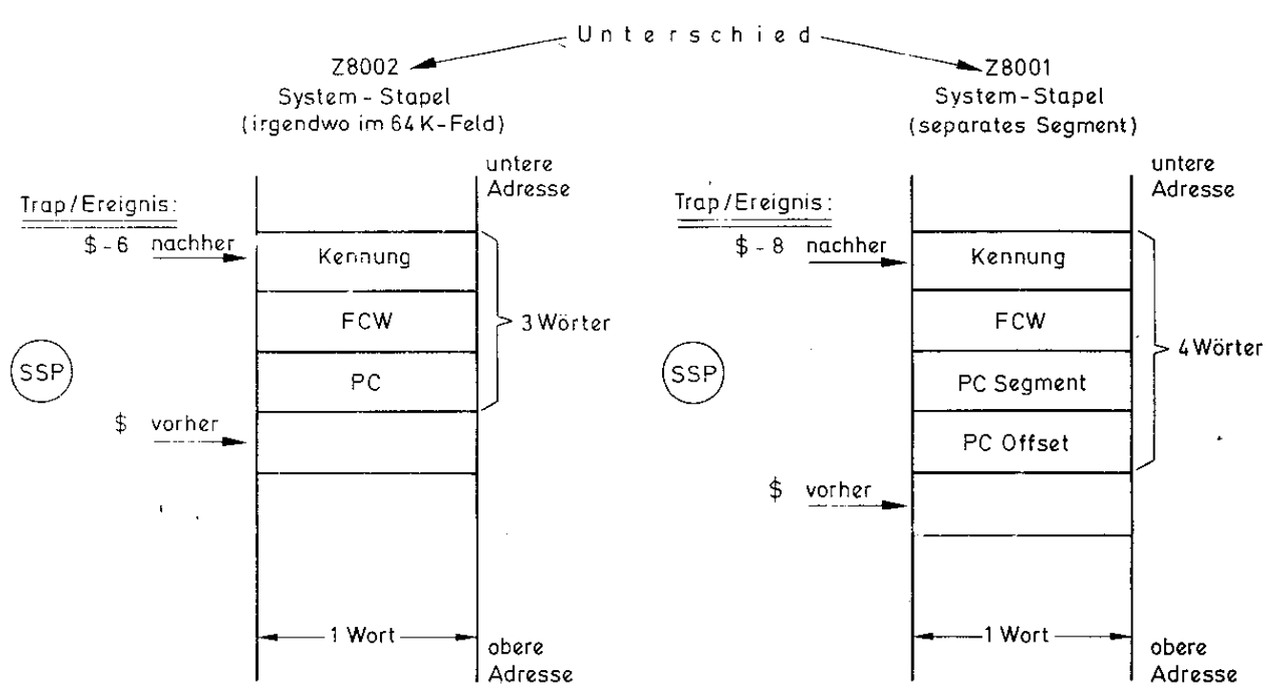
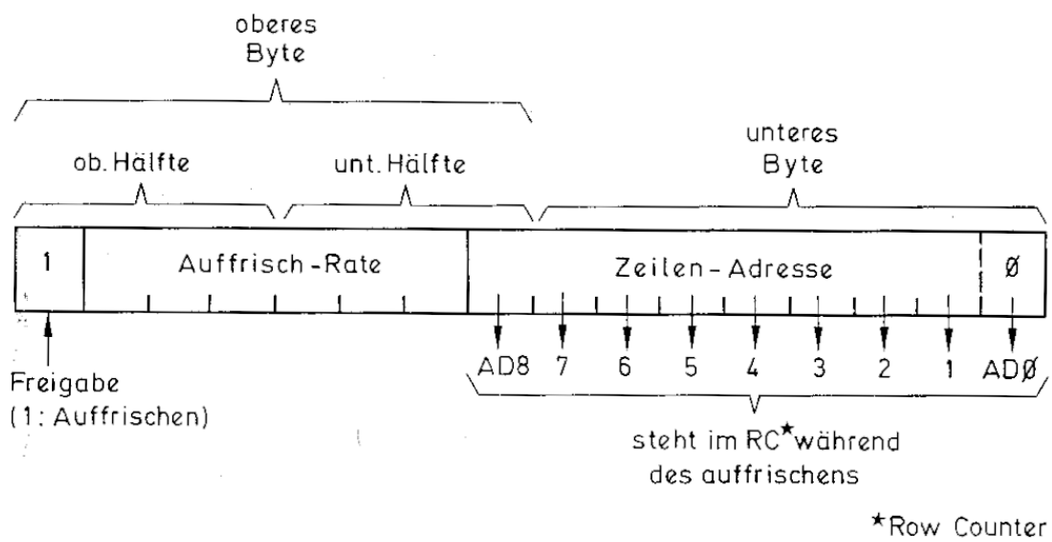


Bild 2.23

Das LSB des Rate-Zählers ist Bit 9. Steht Null im Ratezähler, bedeutet das, daß erst nach 256 Taktzyklen (entsprechend 64 μs bei einem 4 MHz-Takt) ein Refreshzyklus erfolgt. Steht 1 im Ratezähler (L00000L...), erfolgt nach 4 T-Zyklen ein Refresh. Steht 2 darin, ist nach 8 T-Zyklen ein Refresh fällig usw. Wird das Z 8000-Refresh-Register mit dem Wert BD00H programmiert, wird erst nach 248 T-Zyklen aufgefrischt.

Es sei hier nur noch abschließend bemerkt, daß dieses Register, wie auch die vorhergehend beschriebenen Programmstatus-Bereichszeiger, über die entsprechenden LDCTL-Befehle (laden Programmstatus) programmiert werden. Dieser Vorgang ist im Rahmen einer Initialisierung bei einer Einschaltroutine eines Z 8000-Systems zu sehen.



Z8000 Refresh-Register

Rate — untere Hälfte

obere Hälfte

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	kein auffrischen															
7																
8	256		4		8		12		16		20		24		28	
9	32		36		40		44		48		52		56		60	
A	64		68		72		76		80		84		88		92	
B	96		100		104		108		112		116		120		124	
C	128		132		136		140		144		148		152		156	
D	160		164		168		172		176		180		184		188	
E	192		196		200		204		208		212		216		220	
F	224		228		232		236		240		244		248		252	

Bild 2.24

2.6 REGISTERAUFBAU ANDERER 16-BIT-ZENTRALEINHEITEN

Das Thema Z 8000 dieses Buches priorisiert sich zweifellos schon allein darin auf ein bestimmtes Produkt. Das ist auch gut so, denn ein allgemeiner Titel wäre verfehlt, hinter dem eine Konzentration auf ein Produkt steht. Es wäre jedoch andererseits etwas schmalspurig, würde man nur ein Produkt im Auge behalten. In diesem Abschnitt wird eine Gegenüberstellung zu anderen 16-bit-Zentraleinheiten angestellt. Der Begriff Zentraleinheit wurde deshalb gewählt, weil neben vier Einchip-Zentraleinheiten auch zwei Systeme darunter sind, deren Zentraleinheit aus einer mehr oder weniger großen Anzahl digitaler Bausteine unterschiedlicher Dichte besteht. Mit den vier CPUs sind im Sprachgebrauch der Mikrocomputertechnik Einchip-Bauelemente gemeint. Es handelt sich um den 16-bit-Typ TMS 9900 der Firma Texas Instruments, den HMOS-Typ 8086 der Firma Intel, den 16-bit-Bipolar-Prozessor 9440 Microflame der Firma Fairchild und den 16-bit-HMOS-Riesen des Typs 68000 der Firma Motorola. Mit diesen vier CPUs ist im Grunde das Feld genannt, das zu Beginn der 80er Jahre die Größenordnung von Prozessoren repräsentieren, die ausnahmslos das obere Ende der Mikrocomputertechnik darstellt. Mit dem Z 8000 ist dies die Runde der "Großen Fünf". Diese, überwiegend VLSI-Bausteine, gehören zur Creme der Entwicklung in der Mikroelektronik.

Die beiden anderen "diskreten Systeme" werden hier als weiterer Maßstab in die Gegenüberstellung mit einbezogen, weil sie in ihrer Art den Standard der Computertechnik noch immer repräsentieren. Es sind hierzu das System LSI 11/23 von der Firma Digital Equipment und die IBM 370 gemeint, die eine Reihe von Impulsen auf die gegenwärtigen Architekturen der 16-bit-CPUs gegeben haben. Dasnun ein detaillierter Vergleich der genannten Systeme weit über den Rahmen dieses Buches hinausginge, ist von vorneherein eine Einschränkung auf das Thema Register begrenzt worden. Mit der Betrachtung der Registerfelder wird hier in jedem Fall auf ein außerordentlich interessantes Merkmal von Hochleistungsmaschinen Bezug genommen. Wenn hier im weiteren Verlauf der Gegenüberstellung (speziell der CPUs) die bereits genannte Reihenfolge beibehalten bleibt, sollte das nicht primär mit einer Bewertung gleichgesetzt werden. Die genannte Reihenfolge gilt für die zeitliche Verfügbarkeit der CPU. Den TMS 9900 konnte man bereits 1977 käuflich erwerben, wogegen der 68000 erst Anfang 1980 als Einzelstück zu erstehen war. Da nun drei Jahre in der Entwicklung der Mikroelektronik fast ein "Lichtjahr" darstellt, speziell dann, wenn darin noch die letzten 70er Jahre betroffen sind, ist es leicht vorstellbar, daß zwischen dem TMS 9900 und dem 68000 ein beträchtlicher Leistungsunterschied festzustellen ist. Deshalb ist in der oben genannten Reihenfolge wohl auch eine gewisse Reihenfolge der Leistungsklasse zu sehen. Im Rahmen dieser Gegenüberstellung ist es hier weniger angebracht, darüber zu spekulieren, an welcher Stelle der Z 8000 einzureihen ist.

Als erste CPU steht also nun die **TMS 9900-CPU** an, den Kreis der "Großen" einzuleiten. **Bild 2.25** zeigt die internen Arbeitsregister. Der Begriff-Zeiger ist in diesem Fall besser angebracht. Der TMS 9900 hat keine Arbeitsregister im üblichen Sinn. Diese CPU ist eine speicher-orientierte Maschine. Der Arbeitsbereichszeiger (genannt Workspace Pointer = WP) ist dieses Zeigerregister, das im Arbeitsspeicherfeld eines Anwenderprogramms ein Feld von 16 Wortregistern organisiert. Diese externen Arbeitsregister können Operanden eines Befehls und nach erfolgter Ausführung das Ergebnis enthalten. Ein

Speicherplatz als Akkumulator, das ist alles andere als üblich. Diese Speicher-Arbeitsregister enthalten auch Datenadressen oder Indexwerte, wenn gewünscht. Um vor einem Arbeitsregister auf das andere zu zeigen, wird der Bereichszeiger um 2 inkrementiert. Das wird im Rahmen einer Byte-Organisation von Speichern so gemacht. Das Speicher-Arbeitsregisterfeld in Form von 16-bit-Registern ist entsprechend eingeteilt:

Akkumulatoren	(Register 0...10)
Effektive Programmadresse	(Register 11)
Basisadresse	(Register 12)
Inhalt des WP einer vorhergehenden Operation	(Register 13)
Inhalt des Programmzählers einer vorhergehenden Operation	(Register 14)
Programmstatus	(Register 15)

Am Beispiel einer Multiplikation wird gezeigt, wie der TMS 9900 diese Operation behandelt. Das 32-bit-Ergebnis wird in zwei aufeinanderfolgende Speicher-Arbeitsregisterstellen abgelegt. An der Speicher-Zeigerregisteradresse W_{ax} wird das höherwertige Ergebniswort abgelegt. Das niederwertige Wort wird nach $W_{ax} + 1$ transferiert (in die nächsten beiden Adreßbytes). Zuerst müssen jedoch die beiden 16-bit-Operanden unmittelbar an die Eingänge der ALU herangeführt werden. Zu diesem Zweck muß das Zeigerregister diese Werte erst in ein internes Zwischenregister hereinladen. Daran führt kein Weg vorbei. Die ALU des 9900 ist eine Random-Logik, die die Operanden für eine Arithmetik parallel benötigt. Wenn man sich das Diagramm des internen Aufbaus der CPU vornimmt (hier leider nicht abgebildet), wird man nach einem Multiplexer und oberhalb des Programmzählers und Workspace-Registers zwei Register T 1 und T 2 feststellen. Diese Register dienen zum Zweck des Zwischenspeichers.

Wird die MULTIPLY-Operation (Opcode 3800H) ausgeführt, werden also zuerst mit dem WP das erste Wort geholt und mit dem Programmzähler das zweite, wenn es irgendwo an einer Speicherstelle des Programms steht. Der Operand muß von einer Programmstelle erst in das Arbeitsregister (im Speicher) transferiert werden, um von dort aus in die CPU zu gelangen. Dem aufmerksamen Leser kann hier wohl kaum verborgen bleiben, daß der WP einige Male "herumzeigen" muß, bis das Ergebnis in zwei aufeinanderfolgenden Speicher-Arbeitsregistern steht. Der TMS 9900 arbeitet mit einem 2,6 MHz-Takt. Die interne Logik ist in I^2L-Technologie entworfen und arbeitet völlig statisch. Der TMS 9900 benötigt für eine Wort-Multiplikation etwa das Dreifache an Bearbeitungszeit ($\approx 50\,\mu s$), die der Z 8000 dafür benötigt. Der Z 8002A ist in dieser Disziplin vier mal so schnell wie der TMS 9900. Insgesamt ist die Architektur des TMS 9900 für die gegenwärtigen Verhältnisse bereits überholt. Er mußte zwangsläufig von seinen Nachfolgern überholt werden, weil in der genannten Zeitspanne von drei Jahren in der Architekturentwicklung von CPUs so bedeutendes geschehen ist, daß ein Vergleich hier nicht mehr fair wäre. Er war jedoch der erste der Großen. Die Differenz zu 8-bit-CPUs gegenüber dem TMS 9900 ist bereits so drastisch, daß mit diesem ersten 16-bit-Typ im Jahre 1976 unmißverständliche Zeichen für die Zukunft gesetzt wurden.

Wenn man so will, hat die eigentliche Ära der 16-bit-CPUs der Typ **8086** eingeleitet. Es wurde auch treffend in einem Bericht der amerikanischen Zeitschrift "ELECTRONICS" vom 16. Februar 1978 ausgedrückt: "8086 microcomputer bridges the gap between 8- and 16-bit designs". Erstmals wurde in einem Chip die bisher nur von Minicomputern her bekannte Segmentadressierung verwirklicht. Die Erzeugung der physikalischen, 20 bit langen Adresse, wird auf dem Chip erreicht. Die 16-bit-Offsetadresse bildet die maximale Größe eines Segments. Es werden vier verschiedene Typen von Segmenten definiert: Programmcode-Seg-

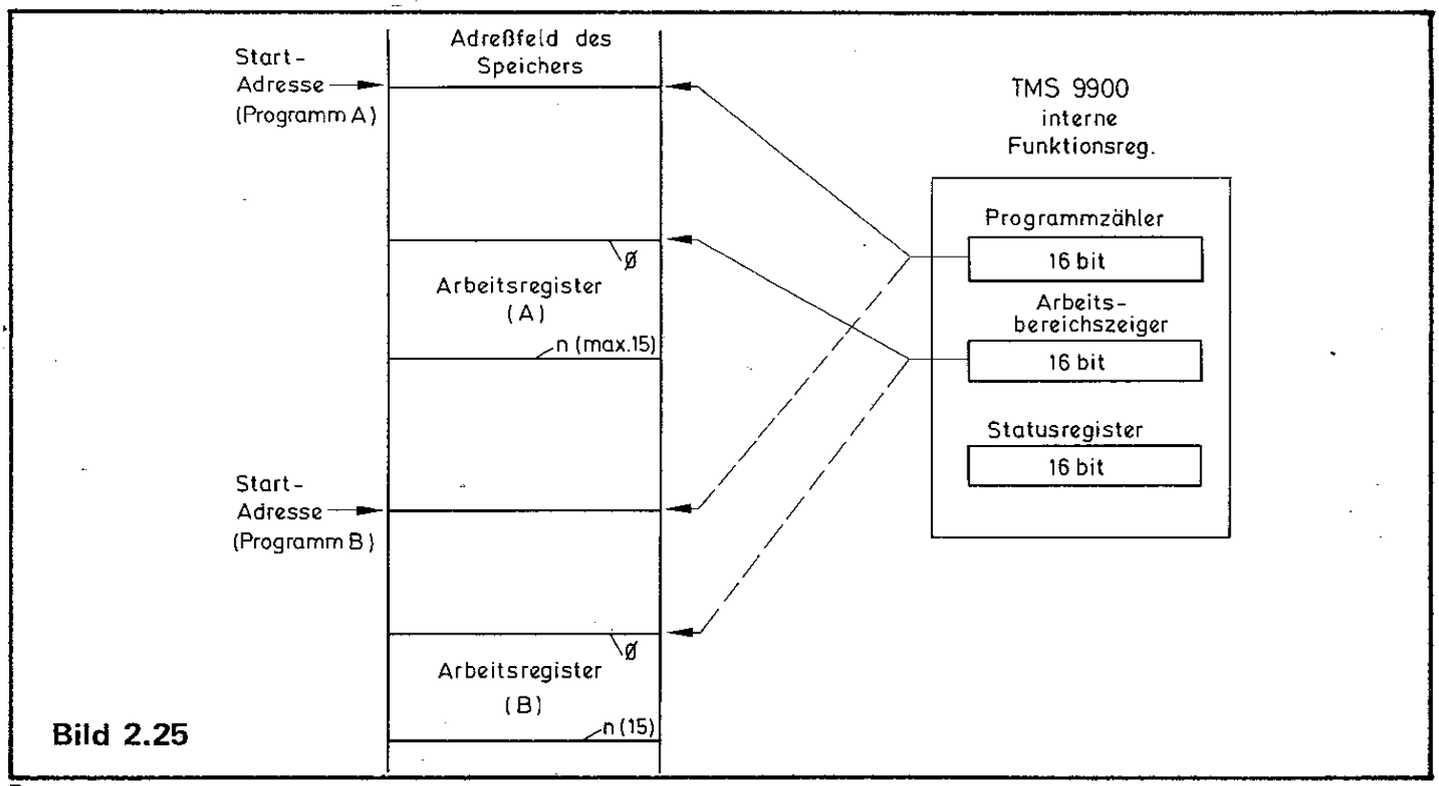


Bild 2.25

ment, Daten-Segment, alternativ variierbares Extradaten-Segment und ein Stapel-Segment. Die Basis-Adresse für ein Segment ist in einem der vier 16-bit-Segmentregister definiert. In **Bild 2.26** ist das Registerfeld des 8086 dargestellt. Diese vier unterhalb der üblichen Register dargestellten Segmentregister bilden die Zeigeregister, deren Inhalt im gesamten adressierbaren Feld von einem Megabyte variabel verschiebbar angelegt werden kann. Somit ist eine Überlappung beispielsweise des Daten- und Extradaten-Segments ohne weiteres möglich. Indem der 8086 bereits eine hardwaremäßige Verknüpfung der Offsetadresse (stehend im Programmzähler) und einer Segmentadresse über ein 20-bit-Addierwerk vornimmt, benötigt diese CPU keine Speichermanagement-Einheit. Die Bildung der physikalischen Adresse für ein demnach organisiertes Speicherfeld ist im Gegensatz zum Z 8001 in einem Chip verwirklicht. Beim Z 8001 wurde dies jedoch bewußt nicht so geplant. Erst die Kombination Z 8001 und Z 8010 bildet rein prinzipiell den 8086. Jedoch besteht zwischen dem 8086 und der Z-Kombination CPU und MMU ein beträchtlicher Leistungsunterschied.

Die Allzweck-Register des 8086 sind differenziert zu betrachten. Die Registergruppe der oberen vier (Akkumulator, Basis-, Zähler-, Datenregister) ermöglichen das Arbeiten mit Operanden, wobei nur der Akkumulator oder das Datenregister als Ziel einer Operation verwendet werden kann. Nimmt man wieder die Multiplikation als richtungsweisendes Beispiel, wird eine 16-bit-Multiplikation mit einem Operanden im Akku und dem anderen in einem direkt adressierten Speicherplatz herangezogen. Diese MUL-Operation ohne Vorzeichen mit 32-bit-Ergebnis erfordert 130 Taktzyklen. Setzt man 200 ns als Taktzyklus bei der 5 MHz-Verrsion an, benötigt der 8086 für die Multiplikation 26 μs. Das Ergebnis steht im Datenregister (höherwertiges Ergebniswort) und im Akkumulator (niederwertiges Ergebniswort). Es muß zum Datenregisterinhalt bemerkt werden, daß das höherwertige Ergebniswort durch mehrmaliges Linksverschieben im Akkumulator und letztlich durch den Transfer vom Akku in das Datenregister zustande kam. Das 16-bit-Datenregister hat keinen Zwischenspeicher als Ergebnispuffer für eine direkte ALU-Verbindung vom Ausgang zum Eingang. Diese Eigenschaft besitzt nur der Akkumulator. Beim Z 8000 hat diese Verbindung jedes Register, ausgenommen der Stacks. Daher ist im 8086-Akku das niederwertige Ergebniswort zu finden.

Es fällt hier des weiteren auf, daß der Z 8000 nicht nur um einige Mikrosekunden schneller ist (18,5 μs, inklusive Operand holen), das sollte man gar nicht so sehr auf die Waage legen; die Problematik liegt dahingehend, daß das schmälere und mit mehr Bedingungen versehene Registerfeld des 8086 zunehmend Transferoperationen erfordert, die mit der eigentlichen Programmieraufgabe nichts zu tun haben. Dies ist speziell dann der Fall, wenn zusammenhängende Operanden bearbeitet werden. Dies kann man überwiegend als die Regel ansehen. Das einfache Z 8000-Programmbeispiel in Abschnitt 2.5 könnte in der gezeigten Form keinesfalls auf den 8086 übertragen werden. Eine Softwarelösung mit dieser CPU würde eine größere Zahl von Transfers erfordern, vor allem bei der Schiebeoperation. Eine Programmentwicklung nach dem Flußdiagramm in Bild 2.15 würde für den 8086 eine etwa 3 mal längere Laufzeit ergeben (Bezug: 4 MHz Z 8000 zu 5 MHz 8086).

Jeweils zwei weitere Register des 8086 sind als Zeiger oder als Indexregister für Operanden einzusetzen (direkte oder relative Adresse). Adreßmanipulationen sind, bezogen auf Zeiger oder Index, möglich. Die Zeigerregister-Inhalte werden als Offsetwert zur Berechnung einer physikalischen Stacksegment-Adresse benutzt. Der Programmzähler-Inhalt dient als Offsetwert zur Berechnung einer physikalischen Adresse in einem Programmcode-Segment.

Zusammenfassend kann über die Architektur des 8086 festgestellt werden, daß eine größere Zahl von Bedingungen während eines Assemblerprogrammentwufes beachtet werden müssen. Eine Gegenüberstellung zur Z 8001-CPU und der Z 8010-MMU ist keinesfalls als Basis anzusehen. Wird eine Gegenüberstellung erwogen, ist vielmehr der Z 8002 zu nennen, der faktisch über ein adressierbares Feld von 384 KByte verfügt (Code-, Daten-, Stacksegment in Normal- und System-Modus). Wie wir bereits kennengelernt haben, ist der Z 8002 nicht segmentiert. Die Unterscheidung in diese "Quasi"-Segmente erfolgt nur über den Status. Eine Gegenüberstellung dieser beiden Prozessoren (im 40poligen Gehäuse) geht hier zumindest von einer vergleichbaren Adreßfeldgröße aus.

Der dritte Prozessor im Kreis der "Großen Fünf" ist die **9440 CPU**. Dieser 16-bit-Bipolarprozessor ist in dem sehr zukunftsträchtigen Isoplanarprozeß I^3L entworfen worden. Diese Maschine lehnt sich softwaremäßig an die NOVA-Minicomputerreihe der Firma Data General an. Diese Maschine besitzt ein mikroprogrammiertes Befehlsdekodierwerk. Die Makrobefehle des NOVA-Befehlssatzes werden über eine Mikrobefehlssequenz erzeugt, dessen vollständige hardwaremäßige Realisierung in einem PLA (programmierbares Logikfeld) eingeschrieben ist. Die Firma Fairchild kann bei den PLAs auf eine sehr breite Erfahrung zurückgreifen. Einige Eigenschaften des 9440 müssen jedoch deutlich genannt werden, die bereits in der vorhergehenden Gegenüberstellung des TMS 9900 und des 8086 angesprochen wurden. Der 9440 ist keine segmentierte Maschine. Er besitzt ein adressierbares Feld von 64 KBytes oder 32 KWorten. Dieser Prozessor unterscheidet nicht zwischen Programmcode-, Daten- und Stack-Segment wie der 8086. Diese Unterscheidung muß der Anwender in seinem Programm selbst definieren. Ein weiterer, fast als schwerwiegend anzusehender Nachteil ist die nur 4-bit-Architektur der ALU. Ein 16-bit-Wort wird im Rechenwerk des 9440 in vier sequentiellen Schritten (4-bit-Wort per Schritt) verarbeitet. Die interne Logik ist völlig statisch ausgelegt, d.h. wie beim TMS 9900 kann die CPU auch mit einzelnen Taktzyklen betrieben werden. Andererseits arbeitet die interne Logik bis zu Tyktzyklen von 83 ns Länge (= 12 MHz Taktfrequenz) noch einwandfrei. Die sich daraus ergebenden Zeitvorteile werden durch die 4-Schritt-Verarbeitung in der ALU wieder zunichte gemacht. Ein Vergleich der Rechenzeiten bei einer 16-bit-Multiplikation ist im Gegensatz zu

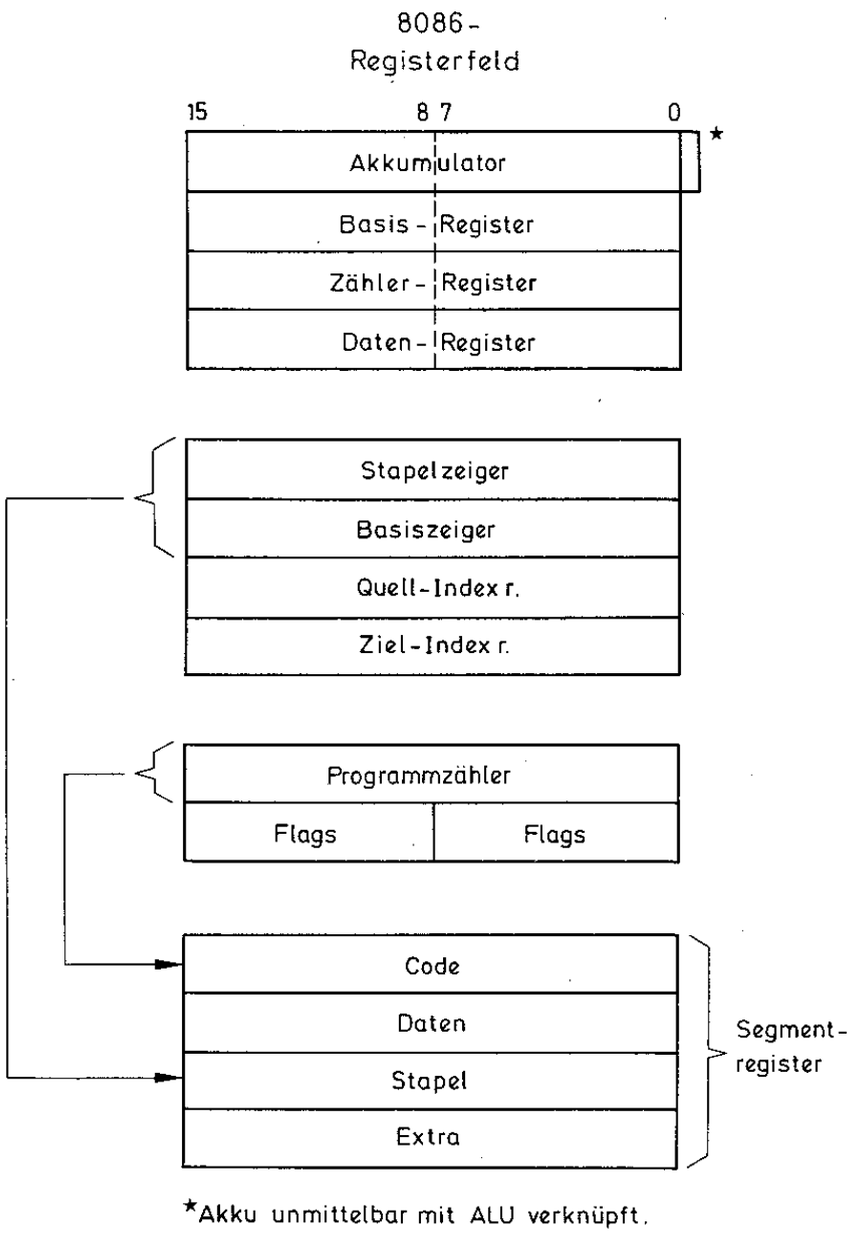


Bild 2.26

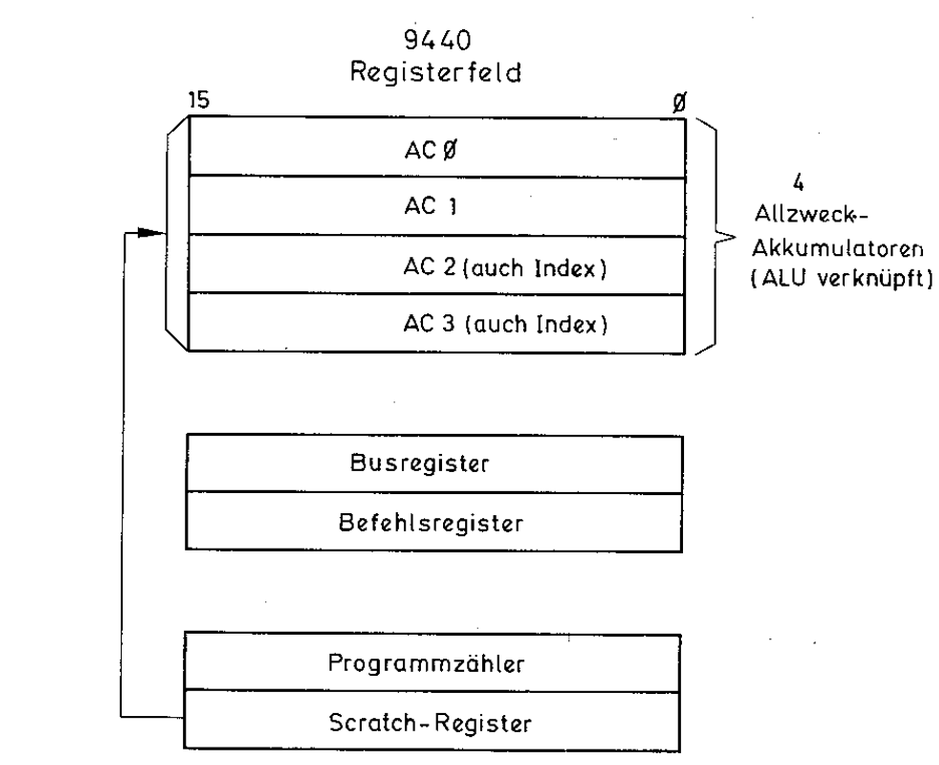


Bild 2.27

den vorangehend beschriebenen CPUs nicht möglich, da der 9440 keinen Einzelbefehl dieser Art besitzt.

Bei der Betrachtung des Registerfeldes in **Bild 2.27** fällt hier doch positiv auf, daß diese CPU vier echte 16-bit-Akkumulatoren besitzt. Was im weiteren diese CPU betrifft, sollte hier keine weitere Betrachtung erfolgen, weil der 9440 in der neuen, "echten" 16-bit-CPU des Typs 9445 einen Nachfolger besitzt. Dieser ebenfalls in I^3L-Technologie entwickelte Prozessor besitzt eine 16-bit-ALU, womit generell das Prädikat "echt" näher bezeichnet ist. Die Akkumulator-Struktur des 9440-Registersatzes wurde beim 9445 um zwei 16-bit-Zeiger erweitert, ein Normal-Stapelzeiger und ein sogenannter Frame-Zeiger. Im Befehlssatz des 9445 sind auch Einzelbefehle für Multiplikation und Division inbegriffen, wobei nach inoffiziellen Angaben für eine 16-bit-Multiplikation 3,5 μs Rechenzeit notwendig sind. Das ist in jedem Fall als sehr schnell zu bezeichnen. Die neue CPU 9445 wird auch keine Segmentierung aufweisen. Sie wird im Laufe des Jahres 1981 zu erwerben sein.

Das jüngste Produkt der Sechzehner-Generation ist der **68000.** Diese CPU verkörpert mit die neuesten Erkenntnisse und Forschungen im Bereich der Architekturen von Einchip-Zentraleinheiten. Sie zeigt in ihrer extensiven Registerstruktur, daß dieser Logikfunktionsgruppe hohe Bedeutung beigemessen wird. Ein universell anwendbarer Registersatz erleichtert das Arbeiten auf Assemblerebene eines Computers ganz beträchtlich. **Bild 2.28** zeigt das Registerfeld des 68000-Prozessors. Die Trennung des Feldes in acht Datenregister und sieben für die Adressen ist keine Einschränkung. Der 68000 hat im Gegensatz zum Z 8000 keinen gemeinsamen Adreß-/Datenbus. Rechnet man noch die beiden Stapelzeiger hinzu, sind alle 17 Register (alle mit 32 bit Länge) auch als Index-Register verwendbar. Dieser Prozessor zeigt in seiner Art eine Extremität in der Halbleiter- und Integrationstechnik. Die Bezeichnung 68000 steht in einem engen Zusammenhang mit der Zahl der integrierten Transistoren auf dem Chip. Nimmt man noch einmal das Programmierbeispiel in Bild 2.15 als Bezug für die Handlichkeit im Umgang auf Maschinenebene, so bearbeitet der 68000 als einziger gegenwärtig existierender Prozessor diese Aufgabe in der gleichen Übersichtlichkeit wie der Z 8000. Statt des Schiebebefehls ist beim 68000 der Rotationsbefehl ROXL einzusetzen. Für die Multiplikation ist der Befehl MULU verwendbar, der zwei vorzeichenlose 16-bit-Operanden zu einem 32-bit-Ergebnis verknüpft. Das Registerbeschreibungsfeld des 16-bit-Befehlsformates gibt das Zielregister an, in welchem das Ergebnis abgelegt wird.

Zweifellos ist der 68000 ein in gleicher Weise außergewöhnliches Produkt wie der Z 8000. Dieser Feststellung gebietet in jedem Fall die Aufrichtigkeit, die dem Thema 16-bit-Prozessoren objektiver Weise zuzukommen hat. Im Kapitel 8 wird zu diesen beiden Extremen der 16-bit-Szene in einer Gegenüberstellung nochmals genauer Stellung genommen.

Hat der Z 8000 einen ideologischen Ursprung? Wenn man diese Frage global stellt, wäre sie falsch gestellt, denn die Architektur des Z 8000 ist aus der Erfahrung seines Schöpfers erwachsen. Das Repertoire an Erfahrung hat hier wesentliche Bereiche des hardwaremäßigen Designs geprägt. Wenn man die Frage in Richtung einer bestimmten Eigenschaft hin betrachtet, haben bestehende Maschinen gewisse Einflüsse ausgeübt. Im ersten Kapitel wurde hierzu bereits Stellung genommen. Würde man die Frage so stellen: "Welches Element grundsätzlicher Art hat bei der Entwicklung des Z 8000 seinen Ursprung aus bestehenden Systemen?", wird man schon eher beim Namen nennen müssen, was hier gemeint ist. Die Produktlebensdauer einer neuen CPU wird in hohem Maße davon geprägt, wie artverwandt ihr Ma-

leichterungen dieser gegenüber seinen Vorgängern bietet. Wenn in den bisherischinenbefehlssatz an einem etablierten Produkt zu messen ist oder welche Er-

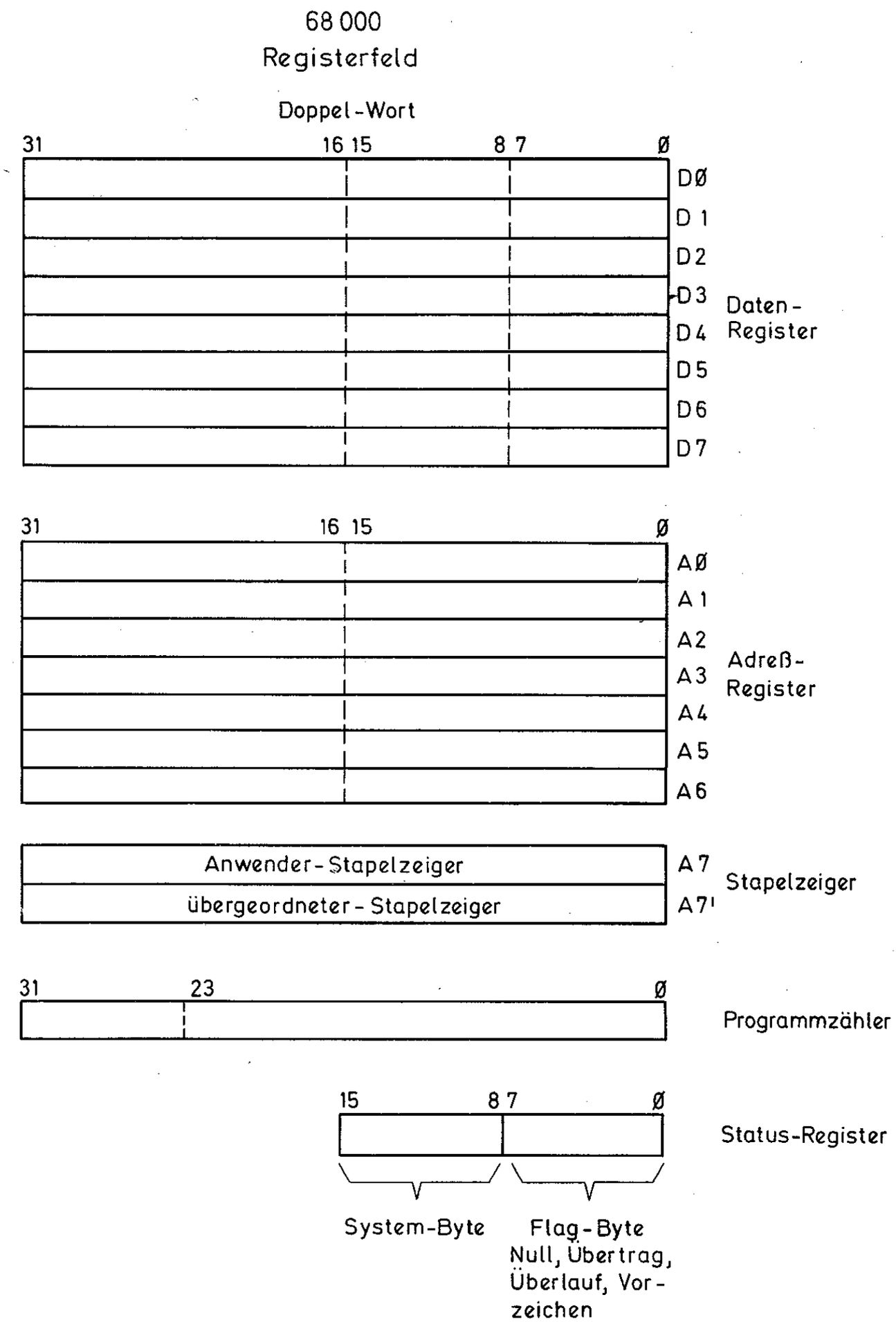


Bild 2.28

gen Ausführungen so gut wie ausschließlich vom Assembler die Rede war, wird im eigentlichen Einsatz die benutzerorientierte Programmiersprache den überwiegenden Raum einnehmen. Der von den Informatikern gerne mit dem Prädikat "syntaktisch arme Struktur" bezeichnete Maschinenbefehlssatz einer CPU ist das Zielobjekt von Compilerentwicklern. Nach ihrer Analyse wird sich zeigen: Hat dieser Maschinenbefehlssatz eine Chance für die Entwicklung eines jeweiligen Compilers der klassischen und modernen Anwenderprogrammiersprachen der höheren Ebene? Sofern nicht das breite Angebot an Maschinen zur Verfügung stand wie heute, war das weniger problematisch. Die Compilerspezialisten mußten sich auf die Objekte (Maschinentypen) konzentrieren, die verfügbar waren. Heute ist das anders. Das Angebot spricht für sich selbst. Werden jedoch für einen Maschinentyp mehrere Compiler entwickelt, ist das gleichsam eine Anerkennung einer CPU auf breiter Basis. Drückt man es einmal sehr konkret auf den gegenwärtigen Zustand aus: Zwischen dem Z 8000 und dem 68000 steht auch die Maschinenideologie des Befehlssatzes, welcher von beiden einem neutralen Compilerentwickler besser in sein Denk- und Entwicklungskonzept paßt.

Sofern man von einer Orientierung beim Konzept des Befehlssatzes beim Z 8000 sprechen kann, ist dieser an den der LSI 11 und der IBM 370 angelehnt. Jeder Kenner weiß, daß auf diesen Maschinen so ziemlich jede höhere Programmiersprache auf einem Cross-Compiler angeboten wird, die Rang, Namen und vor allem Einsatzzuschnitt hat.

Die LSI 11 ist eine auf breiter Basis anerkannte 16-bit-Maschine, die in dem Typ LSI 11/23 eine gewisse Optimierung erreicht hat. Der LSI 11 Zentralbaustein mit einem 40poligen Gehäuse hat zwei Chips aufgesetzt. Der eine Chip wird als Datenchip bezeichnet. Er enthält das Rechenwerk, die 16-bit-ALU, und den Allzweck-Registersatz. Dieses aus 6 Akkumulatoren bestehende Feld ist in **Bild 2.29** dargestellt. Auf dem

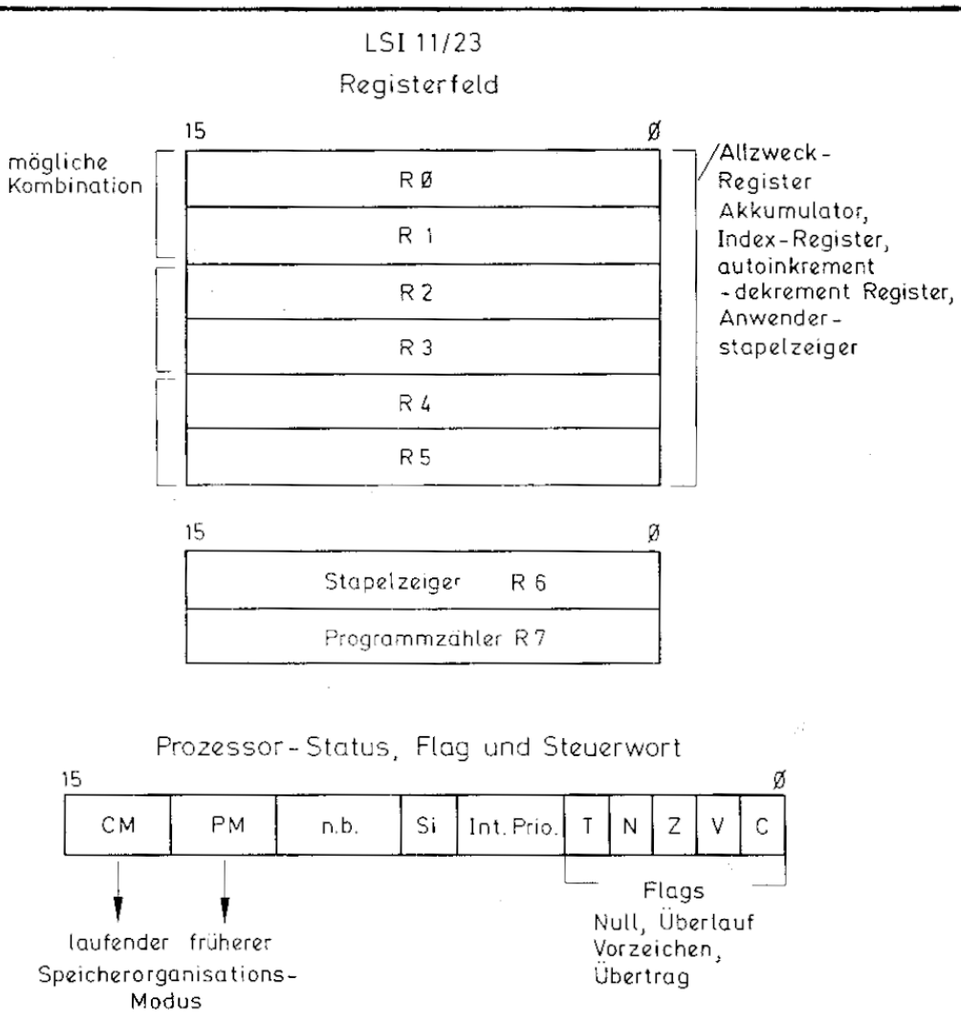


Bild 2.29

Chip befindet sich außerdem noch die Adreß-/Daten-Interfacelogik. Der zweite Chip enthält das Mikroprogramm (in einem Speicher von 552 Worten) und die Sequenzer-Logik. Dieser Chip (Steuerchip) zeigt damit, daß es sich bei der LSI 11 um eine mikroprogrammierte Maschine handelt. Die Memory Management Unit (MMU) ist ein eigener 40poliger Baustein. Sie ermöglicht auch die virtuelle Adressierung in einem physikalischen Speicherfeld. Die Taktversorgung für alle Chips erfordert einen 4-Phasen nicht überlappenden Taktzyklus mit 380 ns typischer Taktzeit. Die LSI 11/23 operiert mit vielfältigen Bedingungen bei der Programmausführung, die in den gegebenen Fällen beim Nichteinhalten Traps verursachen. Dies können ungerade Adressierfehler, Stackfehler, Paritätsfehler des Speichers, falscher Segmentzugriff, Vektoradreßfehler u.a. sein. Systematische Ähnlichkeiten sind zum Z 8000 nicht von der Hand zu weisen.

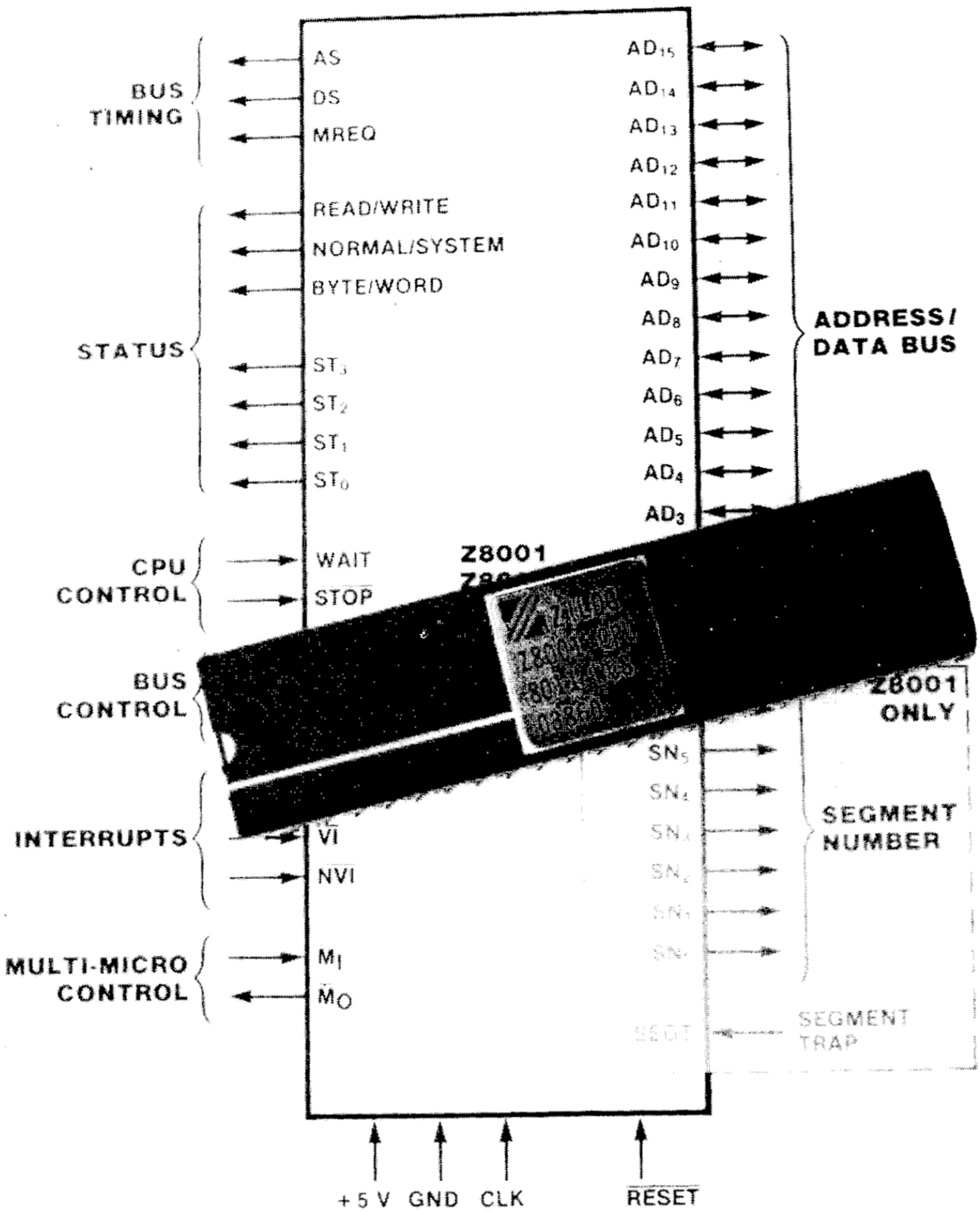
Bei einer vorzeichenlosen Multiplikations-Operation liefert die LSI 11/23 nur dann ein 32-bit-Ergebnis, wenn als Zielregister bei dem Assembler-Syntaxausdruck ein mit gerader Nummer angegebener Akkumulator angegeben wird. In diesem Fall wird in den Akku mit geradzahliger Nummer das niederwertige Ergebniswort abgelegt und in den nächst höheren das höherwertige Ergebniswort. Bei einer ungeradzahligen Zielregisternummer wird in diesen nur das niederwertige Ergebniswort abgelegt. Diese Konvention hat man beim Z 8000 auch zu beachten. Die Rechenzeit bei der Ausführung von Befehlen ergibt sich aus der typischen Zeit für den Befehl und der Zeit für Quell- und Zieltransfer. Die LSI 11/23 benötigt für eine Multiplikation etwa 28 μs. Der Z 8000 ist hier wiederum schneller, etwa 10 μs. Auch mit der LSI 11/23 ist es nicht möglich, das Programmbeispiel nach Bild 2.15 in der gleichen Befehlsfolge wie beim Z 8000 bearbeiten zu lassen. Die LSI 11 hat keinen 32-bit-Schiebebefehl. Sie erfordert einige zusätzliche Transferoperationen.

Der von der Größenordnung her sicher am wenigsten in diese Gegenüberstellung passende Computer ist die **IBM 370.** Ihre Dimensionen sind gegenüber dem Z 8000 einfach zu groß. Das zeigt auch das Registerfeld in **Bild 2.30** dieser Maschine. Die 16 Allzweck-Register mit einer Länge von 32 bit ist ein Indiz dafür. Diese Register können zu acht 64-bit-Registern nach dem gezeigten Schema zusammengefaßt werden. Beim Z 8000 lassen sich immerhin vier 64-bit-Register für Ergebnisse zusammenfügen. Es ist überdies keine Frage, daß die IBM 370 das Z 8000-Programmbeispiel mit der gleichen Effizienz erledigt (ohne zusätzliche Transferbefehle). Auch bei dieser Maschine muß bei der Syntaxbeschreibung des MULTIPLY-Befehls darauf geachtet werden, daß in dem R_1-Feld des 32-bit-Befehlsformats ein Register mit geradzahliger Nummer angeben wird, weil das 64-bit-Ergebnis (mit Vorzeichen) ein Doppelregister beansprucht.

Die IBM 370 multipliziert nur 32-bit-Langwörter mit Vorzeichen. Der Multiplikand wird von dem Register in einem Paar an die ALU gelegt, der eine ungerade Numerierung besitzt. In einem geraden Register steht dann das höherwertige Doppelwort des Ergebnisses, in einem ungeraden des Paares das niederwertige.

Die IBM 370 hat keine Stackregister der üblichen Art. Jedoch erfüllen die Allzweckregister mit entsprechenden Befehlsumschreibungen Anwendungen in dieser Richtung; PUSH und POP gibt es beim 370 nicht.

Betrachtet man den Maschinenbefehlssatz der IBM 370 genauer, muß man zum Teil verblüffende Ähnlichkeiten zum Z-Befehlssatz feststellen. Arithmetische Befehle und Schiebebefehle sind ohnehin ziemlich gleichbedeutend. Der 370er Befehl "Compare (insert) logical character under mask" kommt den Bit-Manipulationsbefehlen BIT, RES und SET des Z 8000 sehr nahe. Ebenfalls sehr nahe kommt der Z 8000-Befehl "Decimal Adjust" dem 370er Befehl


BUS TIMING
AS
DS
MREQ
STATUS
READ/WRITE
NORMAL/SYSTEM
BYTE/WORD
ST3
ST2
ST1
ST0
CPU CONTROL
WAIT
STOP
BUS CONTROL
INTERRUPTS
VI
NVI
MULTI-MICRO CONTROL
MI
MO
Z8001
AD15
AD14
AD13
AD12
AD11
AD10
AD9
AD8
AD7
AD6
AD5
AD4
AD3
ADDRESS/ DATA BUS
Z8001 ONLY
SN5
SN4
SN3
SN2
SN1
SN0
SEGMENT NUMBER
SEGT
SEGMENT TRAP
+5 V
GND
CLK
RESET

"Convert to decimal". Dieser Befehl behandelt in beiden Maschinen die BCD-Arithmetik. Eine besondere Gemeinsamkeit zeigt sich in der weitgehenden Dualität zweier String-Operationen beider Maschinen. Der Befehl TRTDB (Translate and Test) hat ein gleichlautendes Äquivalent eines 370er Befehls.

Dieser kleine Vergleich zeigt hier, daß die IBM 370 bei der Entwicklung des Z 8000-Befehlssatzes Pate gestanden hat. Jetzt erscheint das Einbeziehen dieser großen Maschine in die Gegenüberstellung keineswegs mehr so unbegründet wie es noch zu Anfang den Eindruck vermittelte.

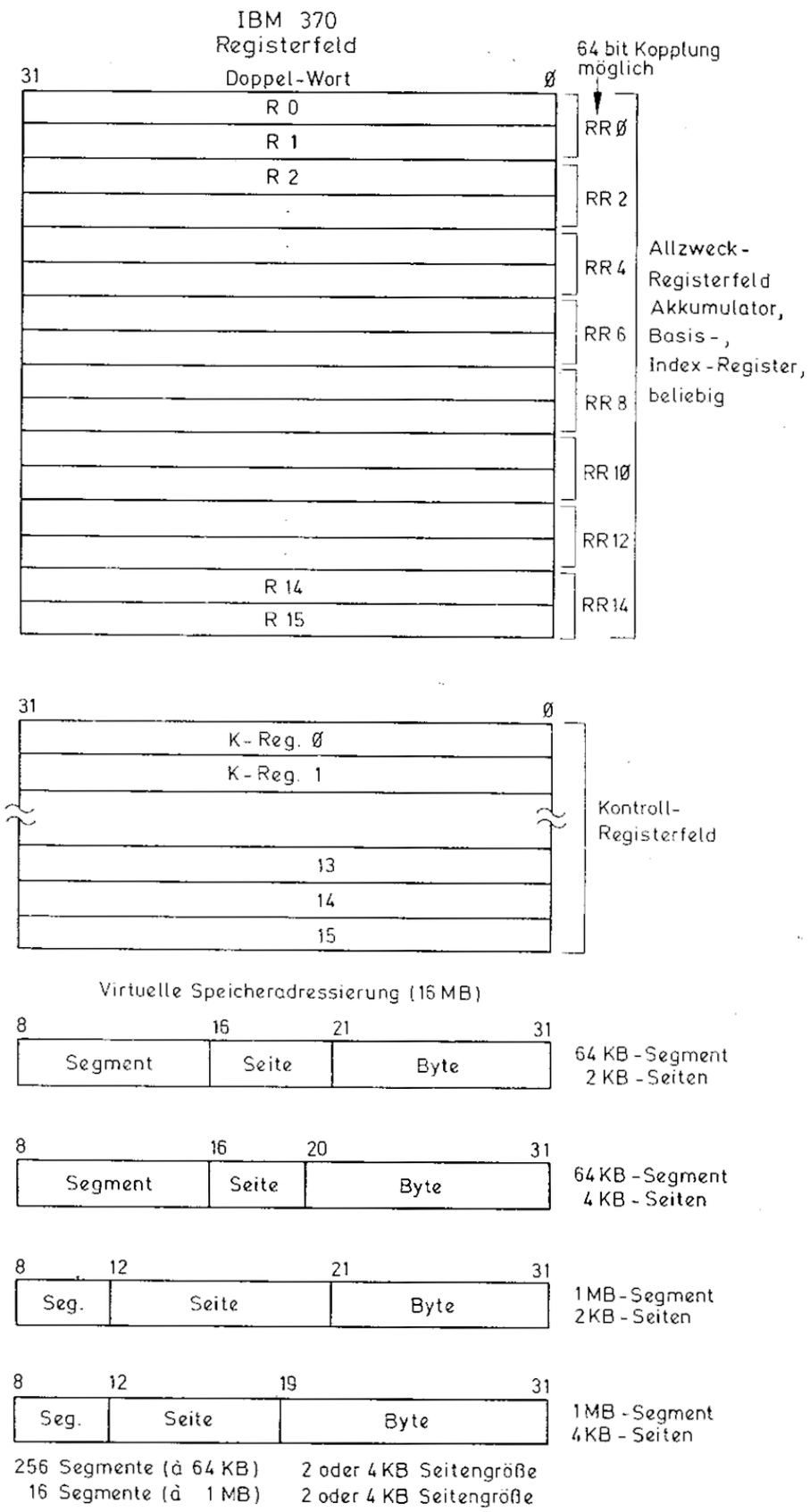


Bild 2.30

2.7 BETRIEBSZUSTANDSANZEIGE WÄHREND CPU-OPERATIONEN

Der Z 8000 verbindet in seiner Eigenschaft als 16-bit-CPU ein kleines DIL-Gehäuse mit einer außerordentlich vielseitigen Statusanzeige. Daß dies so gelöst werden konnte, ist neben dem gemeinsamen Adreß-/Datenbus auf vier binär dekodierte Statusleitungen zurückzuführen. 4 Leitungen zeigen 16 diskrete Zustands- oder Arbeitsphasen der CPU an, von denen 15 eine aktive Bedeutung haben. Hätte man beim Anschlußdesign nicht so streng auf die Effektivität jedes einzelnen Anschlusses geachtet, der Z 8000 müßte zwangsläufig ein 64poliges DIL-Gehäuse besitzen. Jeder Pin des Z 8000 ist ausgereizt bis zum Äußersten. Das Funktionsniveau ist sehr hoch. Kein anderer Prozessor zeigt eine ähnliche Optimierung des Anschlußdesigns wie diese CPU. In diesem Abschnitt konzentriert sich die Beschreibung auf die Anschlüsse ST 0 . . . ST 3 und den N/$\overline{S}$-Anschluß (Normal-System-Betriebsart), die in dem Strukturbild 2.7 zu sehen sind. Der letztere zeigt unmittelbar ein Bit aus dem FCW-Register an:

Die **Statusleitungen ST 0 . . . ST 3** zeigen nach einer entsprechenden Dekodierung jeden nur denkbaren Betriebszustand an, der beim Z 8000 auftreten kann. In der Tabelle der Statusfunktionen ist beschrieben, wie die Permutation der Statusleitungen zu verstehen sind.

Tabelle der Betriebszustände

ST 3	ST 2	ST 1	ST 0	Nr.	Bedeutung
0	0	0	0	0	interne Operation
0	0	0	1	1	Auffrischen des dynamischen RAM-Speicherbereichs
0	0	1	0	2	E/A-Operation mit allgemeiner Anwenderschnittstelle
0	0	1	1	3	E/A-Op. mit der MMU (Befehle SINDR, SINI, SOTIR, SOUT)
0	1	0	0	4	CPU akzeptiert einen Segment-Trap
0	1	0	1	5	CPU akzeptiert einen NMI (nicht maskierter Interrupt)
0	1	1	0	6	CPU akzeptiert einen NVI (nicht vektorisierter Interrupt)
0	1	1	1	7	CPU akzeptiert einen VI (Vektorinterrupt)
1	0	0	0	8	Anforderungszyklus für den Datenspeicher
1	0	0	1	9	Anforderungszyklus für den Stackspeicher
1	0	1	0	10	EPU-Anforderung für den Datenspeicher
1	0	1	1	11	EPU-Anforderung für den Stackspeicher
1	1	0	0	12	Befehlszyklus für n-tes Wort im Code-Speicher
1	1	0	1	13	Befehlsholzyklus für erstes Wort im Code-Speicher
1	1	1	0	14	Transfer-Operation von der EPU zur CPU
1	1	1	1	15	ohne definierten Betriebszustand

Die Permutation des Status ist einfach in diskrete Signale umzusetzen. Man benötigt dazu einen Binärdekoder-Demultiplexer. Hardwaretechnisch gibt es hierfür mehrere Möglichkeiten. Will man alle Statuswerte diskret zur Verfügung haben, ist der Binärdekoder 74LS154 notwendig (LS-Version nur von Valvo verfügbar). Legt man die Statussignale an die Eingänge A . . . D an, können an den Ausgängen 0 . . . 15 die dekodierten Werte abgegriffen werden. In vielen Fällen reichen jedoch die ersten 10 Werte, die noch den Anforderungszyklus für den Stackspeicher beinhalten. Hierfür ist der BCD-Dekoder 74LS42 oder der Am 26LS2537 ver wendbar. Letzterer bietet die Ausgangsfreigabe mit Tri-State-Treibern und einem Polarisierungseingang zur Definition der Parität der Statuswerte (Hi oder Lo aktiv).

Wie aus der Tabelle zu ersehen ist, wird zwischen internen Operationen, E/A-Schnittstellen, den Interrupts und den Anforderungszyklen genau unterschieden. Interne Operationen werden angezeigt, wenn die CPU intern Daten bearbeitet. Dieser Status wird unmittelbar im ersten Taktzyklus des ersten Ausführungszyklusses ausgegeben. Das erfolgt bei längeren Verschiebe- oder Rotationsoperationen in einem Register und insbesondere bei zeitintensiver Arithmetik (MULT, DIV). Bei E/A-Operationen mit einer allgemeinen Schnittstelle, die mit dem Übernahmebefehl IN und dem Ausgabebefehl OUT gemacht wird, zeigt der Status den Wert Nr. 2 an. Wird eine Initialisierung oder Leseoperation mit dem Speicherorganisationsbaustein MMU vorgenommen, ist der Status Nr. 3 aktiviert. Initialisiert oder gelesen werden die zahlreichen Register der MMU mit den Befehlen SOUT, SOUTD, SOUTI, SOTDR, SOTIR, bzw. SIN, SIND, SINI, SINDR, SINIR. Für die Korrespondenz CPU — MMU steht in der Tat eine Befehlsvielfalt zur Verfügung, E/A einfach, dekrement, inkrement und wiederholend.

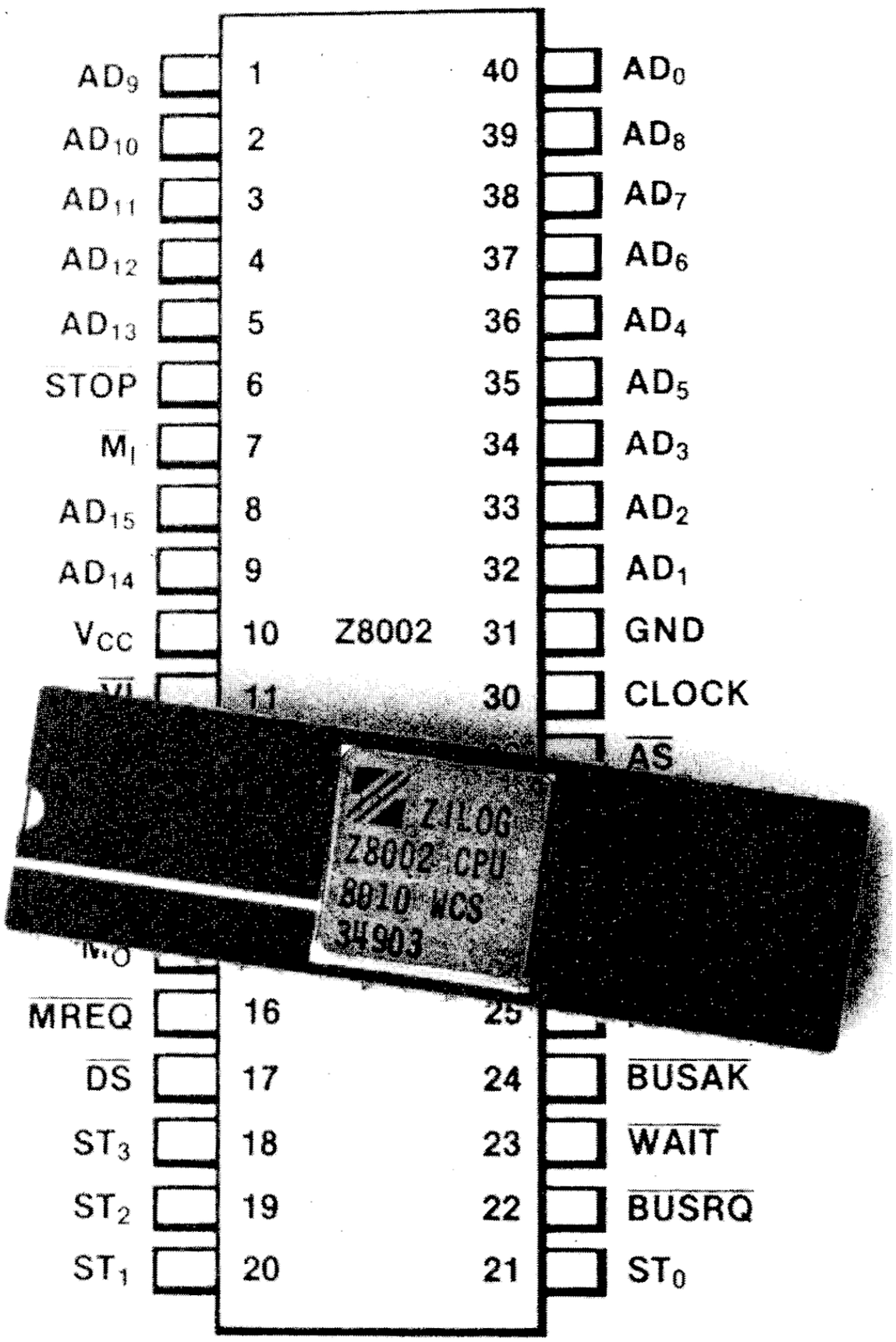
Status Nr. 3 und 4 sind verständlicherweise nur aktiv, wenn diese Operationen mit einem Z 8001 und einer vorhandenen Z 8010-MMU durchgeführt werden. Wie ein Segment-Trap entsteht, wird in Kapitel 5 erklärt. Die Interrupt-Anerkennung ist hingegen für beide CPUs wichtig. Die Anforderungszyklen für Daten und Stack und die Befehlsholzyklen im Programmspeicher helfen Speicherfelder zu segmentieren oder zu unterteilen. Obwohl die Z 8002-CPU nur eine 16-bit-Adresse ausgeben kann, ist sie in der Lage, weit mehr adressieren zu können. Sie trennt durch die Ausgabe des entsprechenden Statusses zwischen Programm-, Daten- und Stackspeicher. Der Status Nr. 12 und 13 wird dann ausgegeben, wenn irgendein Befehl geholt wird, also die Adresse vom Programmzähler kommt. Der Datenspeicher wird angesprochen, wenn in einem Register die Zieladresse eines Operanden steht (indirekte Registeradressierung). Der Stackspeicher wird angesprochen, wenn über CALL oder RET in oder aus dem Stack die neue Programmzähleradresse geholt, geschoben wird, also der Stapelzeiger die Adresse ausgibt. Somit sind die drei Speicherbereiche durch den Status definiert.

	— Programmzähler	→ Status Nr. 12 und 13
Adresse	— Register	→ Status Nr. 8
	— Stapelzeiger	→ Status Nr. 9

Damit ist die Bedeutung der diskreten Statussignale gut zu verstehen. Diese Signale sind überdies sehr schnell aktiv. Nach etwa 100 ns im ersten Takt eines beliebigen Maschinenzyklusses kann man mit einem definierten Status rechnen. Zählt man weitere 20 ns für die Dekodierung hinzu, hat man was man braucht, um ein Inhibit- oder Freigabesignal an Speicher oder Peripherie zu senden.

Um auf den Stapelspeicherbereich zurückzukommen. Dieser Speicherblock kann zu nichts anderem benutzt werden, als Sprungadressen zu speichern (damit man sich hier keinen falschen Hoffnungen hingibt). Theoretisch stehen 64 KB zur Verfügung; jedoch, wer braucht das schon.

Die Statusnummern 10, 11 und 14 zeigen eine Besonderheit des Betriebszustandes der CPU an. Sie zeigen die Zusammenarbeit der Z 8000-CPU mit der Z 8000-EPU an. Im ersten Kapitel war des öfteren von einem Entlastungsprozessor die Rede, der in der Lage ist, der CPU Routine-Transaktionen abzunehmen. Die CPU ist für eine Zusammenarbeit mit diesem Extended Prozessor vorbereitet. Die Bemühungen in diese Richtung hatten das Ziel, die Zusammenarbeit CPU ↔ EPU so unkompliziert wie nur irgend möglich zu gestalten. Die EPU ist auch nicht mit der üblichen und bekannten Koppelstrategie verschiedener Peripherie-Prozessoren vergleichbar. Für den Z 8000 wurden über den bekannten Befehlssatz hinaus eine Reihe von Befehlen entworfen, die für die EPU als ausführendes Organ gedacht sind.


AD9 1
AD10 2
AD11 3
AD12 4
AD13 5
STOP 6
MI 7
AD15 8
AD14 9
VCC 10
Z8002
31 GND
30 CLOCK
AS
ZILOG
Z8002 CPU
8010 WCS
34903
MREQ 16
DS 17
ST3 18
ST2 19
ST1 20
25
24 BUSAK
23 WAIT
22 BUSRQ
21 ST0
AD0 40
AD8 39
AD7 38
AD6 37
AD4 36
AD5 35
AD3 34
AD2 33
AD1 32

Dieser "Extended-Satz" besteht aus sechs Opcodes. Er ist kurz mit EPU intern, Datentransfer EPU-Speicher-CPU und Statustransfer EPU-CPU zu beschreiben. Wird einer dieser Befehle von der CPU über den AD-Bus geholt, hat die EPU gleichzeitig davon Wind bekommen, weil sie in einer Art Beobachterposition durch den direkten Anschluß am AD-Bus sozusagen den Überblick über die Transaktionen der CPU erhält. Der "Extended Befehl" wird von der CPU und der EPU zwischengespeichert. Die CPU fragt das EPU-Bit (Nr. 13) im FCW-Register ab und entscheidet daraufhin über die Ausführung dieses Befehls von der EPU. Wenn durch das gesetzte Bit im FCW die Ausführung dieses Extended-Befehls der EPU übertragen werden darf, zeigt die CPU an den genannten Statusnummern an, daß die EPU den Befehl ausführen darf. Die EPU braucht zu diesem Zweck nicht weiter initialisiert werden, denn sie weiß bereits Bescheid, wenn man so will. Die drei Arten der Statuswerte zeigen der Peripherie an, welche Transaktionen die EPU vornimmt. In Kapitel 10.2 wird auf die Z 8000-EPU noch detaillierter eingegangen. Ist das EPU-Bit im FCW nicht gesetzt, wird von der CPU auch kein Status des genannten Typs ausgegeben. Damit wird der EPU nicht die Genehmigung erteilt, die CPU über die STOP-Leitung (CPU-Eingang) anzuhalten. In der CPU wird in diesem Fall intern ein "Extended Instruction Trap" erzeugt, der sie veranlaßt, in eine Service-Routine zu springen, die die simulierte Bearbeitung eines EPU-Befehls zum Ziel hat. Bei diesem Vorgang wird der Programmstatus-Bereichszeiger PSAP aktiv. Der Status für die entsprechende EPU-Transaktion wird in der schon erwähnten Zeit einer allgemeinen Statusgenerierung erzeugt.

Der Anschluß 31 des Z 8001 bzw. 26 des Z 8002 zeigt hardwaretechnisch eine weitere Besonderheit der CPU an. Die Programmbearbeitung kann im **Normal-** oder **System-Modus** ausgeführt werden. Der Unterschied beider Betriebsarten ist sehr leicht an Hand der einfachen Grafik in **Bild 2.31** zu begreifen. Der Z 8000 hat unter den bereits erwähnten 110 Befehlsgruppen (414 Befehle) einige Opcodes höherer, systempriorisierter Bedeutung. Das soll nicht bedeuten, daß diese Befehle besser, schwieriger oder schneller wären. Im Wesentlichen handelt es sich um Befehle, die zur Initialisierung der Z 8000-Systemkomponenten (inklusive der CPU selbst) oder zur Übernahme bzw. Ausgabe bestimmter Datenblöcke an die Peripherie verwendet werden. Knapp ein Fünftel aller Befehlsgruppen fallen unter dieses Privileg, nur dann vom Z 8000-Anwender in Anspruch genommen werden zu dürfen, wenn im FCW-Register das Bit 14 gleich 1 gesetzt ist. Was bringt die Trennung in zwei Betriebsarten beim Z 8000?

Die CPU wird sowohl in Mehrprozessorsystemen als auch in entsprechenden Ausbaustufen für mehrere Anwender (Multi-user-Betrieb) eingesetzt. In relativ großen Systemen dieser Art werden Dienstprogramme eingesetzt, die im wesentlichen auf das Durchsuchen irgendwelcher Tabellen (im Speicher) ausgerichtet sind. Werden solche Datentabellen nur gelesen, ist das nicht weiter kritisch. Wird gelesen und geschrieben, sollen die Daten der Tabelle keinesfalls blind überschrieben werden können. Eine vorhergehende, genaue Definition eines Tabellen-Datensegments soll verhindern, daß Daten blind überschrieben werden (z.B. wegen falscher Bedienung durch Personal) und darüber hinaus nicht jeder Benutzer einer solchen Datentabelle die Befugnis erhält, die Definition eines Datensegments willkürlich zu ändern. Sind diese Dienstprogramme im Normal-Modus geschrieben, ist dafür ein Schutz vor dem genannten Fall gegeben. Ein Datensegment umzudefinieren, ist nur im System-Modus möglich. Durch entsprechende Initialisierung der MMU kann ein bestimmtes Datensegment nur gelesen oder höchstens erweitert werden, aber nicht überschrieben. Die Initialisierung der MMU (dieser Baustein wird in größeren Systemen verwendet) ermög-

licht eine sehr spezifische Beschreibung eines jeweiligen Segments. Diese Initialisierung kann nicht jeder Anwender vornehmen. Die Initialisierung ist nur möglich, wenn in einem Terminal ein System-Mode-Zugriff erlaubt ist. Eine mainframe-Initialisierung wäre beispielsweise dann leicht durchführbar, wenn in jedem Terminal ein Z 8002-Steuerprozessor eingesetzt würde, der eine Systeminitialisierung an das Z 8001-Hostsystem zuließe, wenn der Steuerprozessor im System-Mode arbeitet. Das Hostsystem könnte immer im System-Mode betrieben werden, wobei die Terminals (ausgenommen von einem oder zwei) nur Normal-Mode-Programme aufrufen können. Über ein oder zwei Terminals kann eine mainframe—Initialisierung durchgeführt werden. Die Terminals können überdies als eigene Z 8002-Systemcomputer ausgebildet sein, so daß im Hostsystem eigentlich gar kein kompliziertes Multiuser-Betriebssystem enthalten sein müßte. Auf Anforderung über das Subsystem-Terminal kann das ganze Programm plus Datenblöcke in das Subsystem transferiert werden. Der Anwender kann dann das Programm benutzen, Daten manipulieren, ohne daß im Hostspeicher irgendein "Schaden" angerichtet wird. Dieses Beispiel beschreibt jedoch schon eine höhere Systemphilosophie eines großen Z 8000-Systems. Der Vorteil würde darin liegen, daß man sich einigen Aufwand an Betriebssystem-Software sparen könnte. Was man sich an Hardware erkauft, kann an einem teueren Ende gespart werden.

Eine weitere Möglichkeit des System-Modus liegt in der Simulation von Programmen. Wird ein Programm mit Testdaten im Normal-Modus geschrieben, das sich nur im System-Halbleiterspeicher befindet, kann man leichter eine Fehlersuche angehen, wenn nicht von vorneherein schon über das DOS-mainframe-Betriebssystem gearbeitet wird. Das DOS-Mainframe (Disk-Operating-System) des Host-Computers wird logischerweise nur im System-Modus betrieben, weil es E/A-Operationen enhält. Die Fehlersuche nach erfolgreichem Abschluß eines "Run" im Normal-Modus kann sich beim folgenden Einsatz mit Disk-Daten auf diesen Bereich konzentrieren. Das anfänglich im Normal-Modus entwickelte Programm wird im System-Mode gefahren, weil im Ablauf zu verarbeitende Daten von einer Diskette geholt werden. Läuft das Programm nur mit Testdaten im RAM, weiß man, wo man den Fehler zu suchen hat. Es kann dann nur an der Disk-Datenübertragung oder dem entsprechenden Aufruf liegen. Bei anderen Minicomputern spricht man von einer "Supervisor"-Betriebsart im Gegensatz zum System-Mode.

Der Aufruf von UPROs über CALL wird oder kann mit dem Normal-Stapelzeiger geschehen, wogegen eine Interrupt-Routine ausschließlich über den System-Stapelzeiger aufgerufen wird. Damit wird sehr sinnreich zwischen Normal- und Systemprogrammen unterschieden.

Wird ein priviligierter Befehl im Normal-Modus ausgeführt, entsteht ein Trap. Ist ein Trap aufgetreten, muß er von der CPU nach den entsprechenden Regeln behandelt werden. Dieser Trap versetzt die CPU automatisch in den System-Modus. Die Rettung des momentanen Programmstatus erfolgt in den System-Stapelspeicher, wie in Bild 2.23 bereits dargestellt wurde. Der neue Programmstatus der Trap-Service-Prozedur wird vom Programmstatus-Bereichszeiger geladen.

Es gibt im Z-Befehlssatz einen Befehl, der einen geregelten Übergang vom Normal- in den System-Modus ermöglicht. Dieser Befehl wird SC (System Call) bezeichnet. Unabhängig vom vorhergehenden CPU-Status wird der momentane Programmstatus (PC-Wort und FCW-Inhalt) über den System-Stapelzeiger in den System-Stapelspeicher gerettet. Im 8-bit-Quellfeld (src) muß im Einwort-Format des Befehls angegeben werden, wo der neue Programmstatus aus dem Stapelspeicherfeld geholt werden soll. An dieser Stelle steht dann auch der Programmzählerwert. Im Bild 2.22 wurde dargestellt, wie dieser 8-bit-Sprung-

wert für den Z 8001 zu setzen ist. Für den Z 8002 ist z.B. 12H einzusetzen. Es kommt eben darauf an, wie der Speicher im Programmstatusbereich organisiert ist. Der SC-Befehl wird wie eine Interrupt- oder Trap-Prozedur behandelt, also unter dem Einsatz des PSAP.

Zum System-Modus sei bemerkt, daß ein Anwender eines Z 8000-Platinensystems mit Schnittstellen von vorneherein im System-Modus arbeiten sollte.

Der Anschluß 32 des Z 8001 bzw. 27 des Z 8002 zeigt der Speicherdekodierlogik an, ob die CPU einen Byte- oder Worttransfer vornimmt. Wird ein Befehl geholt, so wird der Programmzähler immer um 1 inkrementiert. In diesem Fall ist der Ausgang B/$\overline{W}$ gleich Hi. Wird eine gerade Adresse ausgegeben (Adreßleitung A 0 = 0), so wird das höherwertige Byte geholt, anschließend um 1 inkrementiert und das niederwertige Byte geholt. Anders ist es, wenn der Inhalt eines Registers die Adresse eines Operanden angibt. Dann wird ein Wortzugriff (B/$\overline{W}$ gleich Lo) ausgeführt. Dieses Signal gibt auf diese Weise genau an, welche Art des Zugriffs momentan erfolgt. Das Signal R/$\overline{W}$ zeigt die Richtung des Transfers an.

Damit ist der Komplex Status des Z 8000 angesprochen worden. In den weiteren Abschnitten und Kapiteln dieses Buches wird auf diese elementaren Zusammenhänge immer wieder zurückgegriffen.

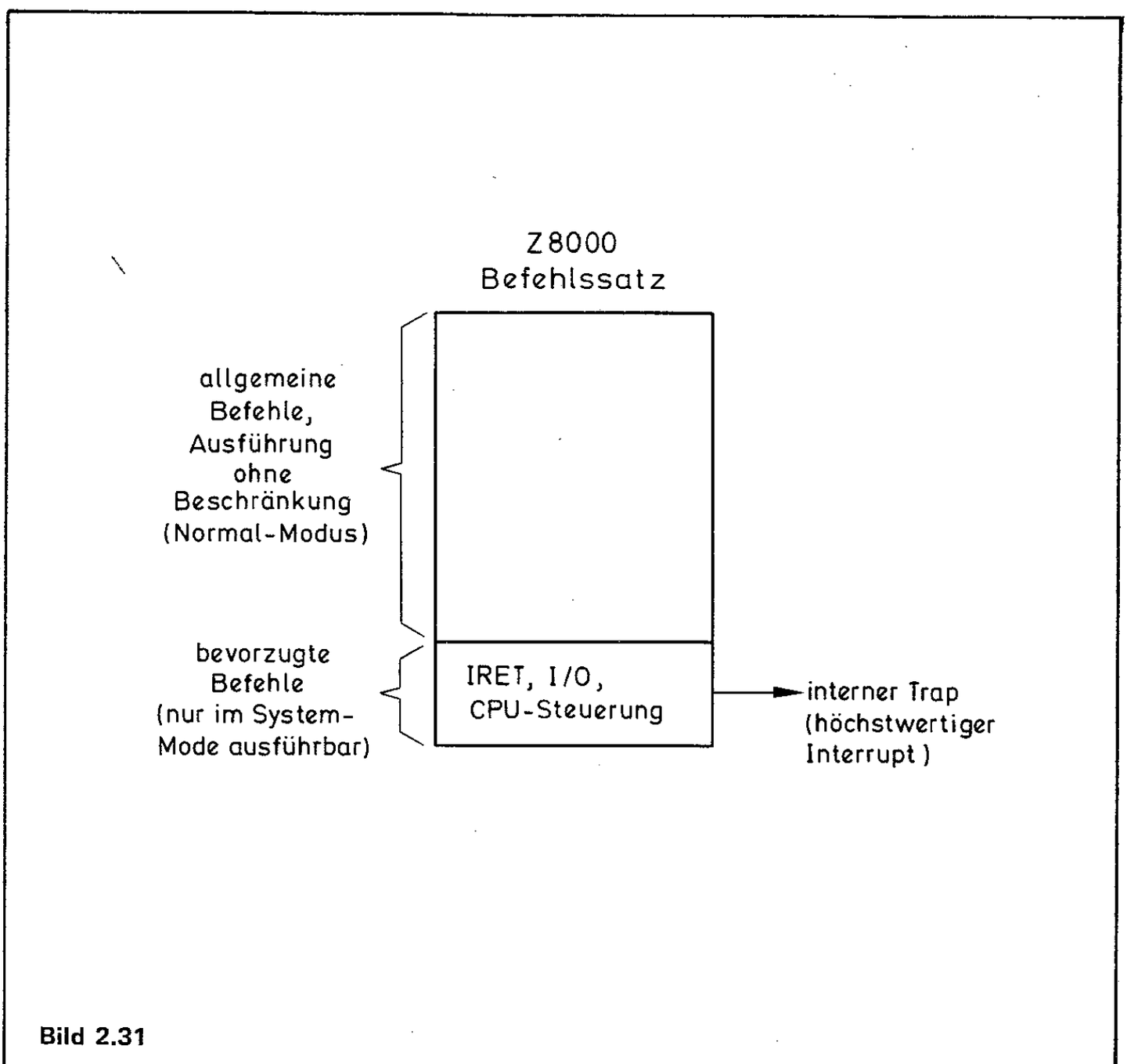


Bild 2.31

2.8 HARDWAREFUNKTIONEN DES Z 8000-BAUSTEINS, SIGNALDIAGRAMME BEI UNTERSCHIEDLICHEN BETRIEBSZUSTÄNDEN

In den folgenden Abschnitten des Kapitels 2 wird dargestellt, was an den Anschlüssen des Z 8000 geschieht, wenn er in Aktion ist. Wie bereits mehrfach angegeben wurde, hat diese CPU einen gemeinsamen Adreß-/Datenbus. Es gibt immer noch Entwickler von Mikrocomputer-Hardware, die vor solch einer Anschlußtechnik zurückschrecken, weil ihnen nicht geheuer ist, daß Adressen- und Datenausgabe absolut getrennt verlaufen. Diese Unsicherheit sollte man schleunigst ablegen, denn sie ist in der Tat unbegründet. Schon der PDP-8-kompatible 12-bit-CMOS-Prozessor des Typs 6100 hat einen gemeinsamen Adreß-/Datenbus, der zeitlich eindeutig gesteuert ist. Beim 8085 von Intel ist das niederwertige Adreßbyte ebenfalls mit dem Datenbus zusammengelegt. Die Adresse kann eindeutig durch den ALE-Ausgang von den Daten unterschieden werden. Beim 8086 sind die Verhältnisse noch ähnlicher zum Z 8000.

In **Bild 2.32** wird an dem elementaren Zeitdiagramm dargestellt, wann die Adresse und wann die Daten am AD-Bus anliegen. Die entscheidenden Zeitsteuersignale zur Definition der Adresse und der Daten sind die Ausgänge Adreß-Strobe ($\overline{AS}$) und Daten-Strobe ($\overline{DS}$). Diese Signale zeigen einen absolut gültigen Zustand der jeweilig am AD-Bus liegenden Signalverknüpfung an. Die angegebenen Zeiten gelten für einen Taktzyklus von 250 ns. Die Lo↑Hi-Flanke an $\overline{AS}$ zeigt eine absolut gültige Adreßausgabe an. Bis zur Lo↑Hi-Flanke an $\overline{DS}$ werden gültige Daten vom Bus in die CPU übernommen. Bei der Datenausgabe gilt der gleiche Flankenzusammenhang. Man sieht überdies, daß die CPU für einen Holvorgang 3 T-Zyklen benötigt.

Wenn man sich **Bild 2.33** betrachtet, kann man die Sequenz ersehen, wie ein Befehlshol- und Ausführungsablauf beim Z 8000 vor sich geht. Entsprechende Holzyklen werden von einem Auffrischzyklus für die dynamischen RAMs abgelöst, sofern im Refreshregister Bit 15 (Refresh-Freigabe) auf 1 gesetzt ist. Der erste Befehlsholzyklus (Instruction-fetch 1 = IF 1) kann bis zu 10 T-Zyklen dau-

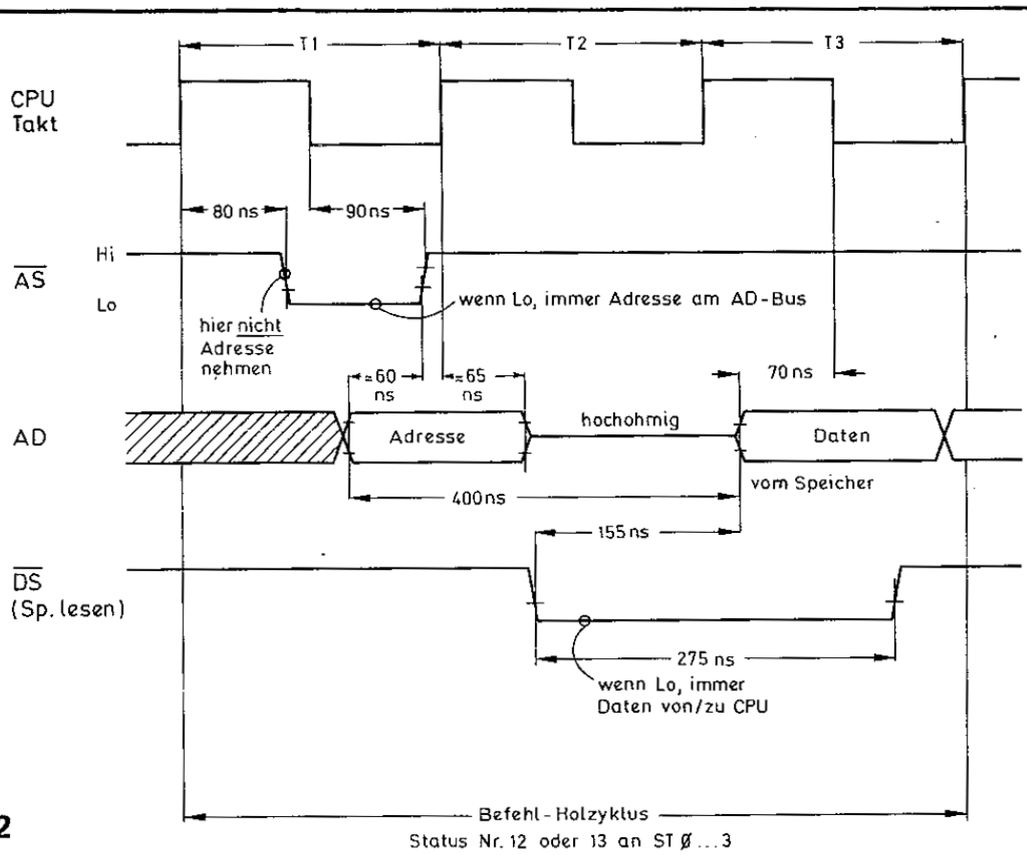


Bild 2.32

ern. Spätestens dann wird aufgefrischt. Der jeweilige Status zeigt an, was die CPU momentan macht. Die CPU läßt sich (auf gut Deutsch) schon sehr genau "auf die Finger schauen", welche Operation sie momentan ausführt.

In **Bild 2.34** werden die beiden Kernoperationen der CPU detailliert gezeigt, den **Lese-** oder **Schreibvorgang** mit dem **Speicher.** $\overline{AS}$ ist im ersten CPU-T-Zyklus aktiv. Zugleich wird im Lo-Abschnitt des ersten Taktes das Signal $\overline{MREQ}$ aktiv, was besagt, daß es sich um eine Speicheroperation handeln muß. Nachdem im Lesezyklus eine Adresse ausgegeben wurde, ist sie im zweiten T-Zyklus wieder inaktiv. Der Bus wird hochohmig. Die CPU fragt im zweiten T-Zyklus den $\overline{WAIT}$-Eingang ab (genau an der Hi↑Lo-Flanke des Taktes) und entscheidet daraufhin, ob zwischen dem zweiten und dritten T-Zyklus ein Wartezyklus eingefügt werden muß. Wartezyklen sind notwendig, wenn zu langsame EPROMs oder andere Typen im Festwertspeicherfeld eingebaut sind. Rechnet man alle Verzögerungen an Zwischenspeichern, Buffern und der Speicherdekodierlogik hinzu, **müssen** die Speicher ab der Lo↑Hi-Flanke von $\overline{AS}$ in 290 ns den Inhalt des adressierten Speicherplatzes auf den Bus gegeben haben, sonst gibts Ärger. Die CPU würde andernfalls irgendetwas auf dem Bus nehmen. Schiebt man einen Wartezyklus (WZ) ein, ist gesichert, daß auch langsame Speicher beim schnellsten Z 8000-Takt betrieben werden können. Sehr schnelle Speicher benötigt man beim Z 8000A (6 MHz-Takt), wenn kein WZ eingefügt werden soll. Hier müssen die Speicher in weniger als 200 ns den adressierten Inhalt herausgeben.

Ob die CPU ein Wort oder ein Byte nimmt, hängt von B/$\overline{W}$ ab. Ist der B/$\overline{W}$-Ausgang der CPU gleich Hisund A 0 gleich Null (gerade Adresse), übernimmt die CPU das höherwertige Byte D 8 . . . D 15 vom Bus. Beim nächsten Inkrementieren ist A 0 gleich 1 (ungerade Adresse). Hier nimmt die CPU das niederwertige Byte D 0 . . . D 7. Man sollte hierzu noch anmerken, daß dieses Nehmen der Daten von der CPU noch durch äußere Hardware unterstützt werden muß. Die Bustreiber an den Speichern müssen so geschaltet sein, daß genau bei der geraden Adresse das obere Byte vom Speicher freigegeben wird, bei der ungeraden Adresse das untere. Das jeweilig andere Byte muß durch den hochohmigen Zustand dieses Busteils nicht übertragen werden können. Wie aus dem Bild 2.34 zu ersehen ist, hat die CPU beide Bytes auf Empfang geschaltet. Nur das zur jeweiligen Adresse passende Byte darf auf den AD-Bus freigegeben werden. Im Wort-Modus ist das unerheblich. Hier muß die CPU beide Bytes am AD-Bus vorfinden. Beim Schreiben gibt die CPU sowohl auf D 0 . . . D 7 als auch auf D 8 . . . D 15 das gleiche Byte aus, wenn im Byte-Modus gearbeitet wird. Je nachdem, wie die Speicherdekodierung ausgelegt ist, wird das Byte in den Speicher für das obere Byte oder untere Byte geschrieben. Beim Schreiben im Wort-Modus werden beide Bytes gleichzeitig in den Speicher geschrieben. Das Umschalten von der Adresse auf die Daten erfolgt in etwa 20 ns. Die jeweiligen $\overline{AS}$/$\overline{DS}$-Flanken beim Lesen werden auch beim Schreiben benützt. Die Hi↓Lo-Flanke von $\overline{DS}$ kann überdies auch als $\overline{CAS}$-Impuls (Zeilen-/Spaltenumschaltung der Adresse bei dynamischen RAMs verwendet werden. werden.

In **Bild 2.35** ist die zweite wichtige Operation des Z 8000 signaltechnisch dargestellt. Wie aus der Inschrift der Adresse zu ersehen ist, sollte diese einen Port oder eine allgemeine Schnittstelle adressieren. Für einen allgemeinen I/O- oder E/A-Port gibt es eine breitere Definition. Grundsätzlich ist darunter ein Baustein zu verstehen, der den Computer mit irgendeiner Peripherie verbindet bzw. über dessen Logik parallel oder seriell Daten ausgetauscht werden können. Jedoch, E/A kann auch ein Speicher sein. Man muß ihn nur so definieren. Es kann ein E/A-Speicher sein (RAM), der per DMA-Betrieb Daten nach außen abgibt,

oder von außen geladen wird. Die Service-Routine des direkten Speicherzugriffs geht dann die CPU nichts mehr an. Ein sehr drastisches Beispiel dieser Art wird in Kapitel 6.5 dargestellt. Das Analog-Datenerfassungssystem lädt die digitalisierten Werte in ein Puffer-RAM, der als Eingabe-Pufferspeicher dient. Als E/A kann des weiteren auch der Z 8038 FIO (128 x 8 bit FIFO-Speicher) angesehen werden. Von der CPU aus gesehen ist dieser Baustein für einen völlig asynchronen E/A-Betrieb ausgelegt.

Der entscheidende Unterschied zwischen Port- und Speicherzugriff ist, daß eine E/A-Operation mit den priviligierten Befehlen IN, OUT oder den speziellen I/O-Befehlen für die MMU programmiert werden muß. Auch bei anderen Prozessoren ist das so. Die Reaktion des Z 8000 ist logisch. Er antwortet auf einen E/A-Befehl mit dem Status Nr. 2 oder 3 und liefert am Ausgang $\overline{MREQ}$ ein diskretes Hi. Ansonsten ist eine E/A-Operation einer Speicher-Operation sehr ähnlich. Jedoch, etwas muß man doch noch beachten. Der Wartezyklus wird bei einer E/A-Operation völlig automatisch zwischen dem zweiten und dritten T-Zyklus eingefügt. Die CPU macht das einfach so, gleich welcher Wert am WAIT-Eingang liegt. Ein WZ wird bei E/As **immer** eingefügt. Ein zweiter wird folgen, wenn $\overline{WAIT}$ sich so verhält, wie in **Bild 2.36** dargestellt ist. Ist $\overline{WAIT}$ zum Zeitpunkt der Hi↓Lo-Flanke während des automatischen WZ gleich Lo, wird nach diesem noch ein WZ eingeschoben. Die CPU benutzt bei einer E/A-Operation grundsätzlich eine 16-bit-Adresse. Die Zeit zwischen der Lo↑Hi-Flanke von $\overline{AS}$ und der Hi↓Lo-Flanke von $\overline{DS}$ dient zur Erzeugung des Freigabesignals ($\overline{CS}$) eines adressierten Bausteins.

In **Bild 2.37** wird der Verlauf eines Auffrischzyklusses für dynamische RAMs dargestellt. Im 6-bit-Auffrischrate-Zähler wird der zu Beginn einprogrammierte Wert mit dem vierten Teil des CPU-Taktes dekrementiert. Hat dieser Zähler den Wert Null erreicht, wird die Zeilenadresse im 9-bit-Zeilenzähler als Adresse ausgegeben. Anschließend wird der Zeilenzähler um 2 inkrementiert. Gleichzeitig wird der programmierte Ratewert neu geladen und es wird wieder aufs neue zum nächsten Auffrischzyklus hingezählt. 1 . . . 64 µs als Auffrischrate sind möglich, wenn die CPU mit einem 4 MHz-Takt betrieben wird. Die Auffrischrate muß dem Typ der dynamischen RAMs angepaßt werden, die in dem entsprechenden Z 8000-System als Schreib-/Lesespeicher verwendet werden. Die bekannten 16 K x 1 dynamischen RAMs erfordern 128 Zeilenrefreshadressen. Der gesamte Refresh muß in 2 ms abgeschlossen sein, wenn man keinen Datenverlust riskieren will. Es ist somit empfehlenswert, die Auffrischrate nicht länger als 15 µs zu wählen, um die 2 ms einzuhalten. Auch bei 64 K x 1 dynamischen RAMs ist mit 2 ms Auffrischzeit zu rechnen. Ist das Refresh-Register des Z 8000 programmiert, braucht man sich um nichts mehr zu kümmern, weil das Auffrischen der RAMs in dem gezeigten Zyklus völlig automatisch abläuft. Der Auffrischzyklus dauert immer 3 CPU-T-Zyklen. Daten erscheinen am AD-Bus nicht, was sich auch durch $\overline{DS}$ gleich Hi konstant zeigt. Der Status Nr. 1 ist aktiviert. $\overline{WAIT}$ bleibt unbeachtet. Eine Auffrischadresse ist immer gerade, weil A 0 immer Null ist. Der Zeilenzähler wird immer um 2 inkrementiert. Hardwaremäßig muß jedoch etwas beachtet werden. **Keinesfalls** darf die **Bussteuerung** z.B. eines **DMA-fähigen Bausteins** einen **Auffrischzyklus behindern.** Das ist einzuhalten.

Ein sehr wichtiger Zyklus ist auch die Anerkennung eines Interrupt oder Trap. **Bild 2.38** zeigt den Zusammenhang. Die CPU arbeitet noch an der Ausführung eines Befehls oder sonstwo. Erscheint ein $\overline{NMI}$ (nicht maskierter I.) am CPU-Anschluß, muß er mindestens 100 ns aktiv bleiben (Lo), bis das interne $\overline{NMI}$-Flag (steht nicht im FCW) gesetzt ist. Der nächste Befehl wird noch geholt (Status Nr. 13), bis dann der Anerken-

Z8000 Befehlshol-und Ausführungsablauf

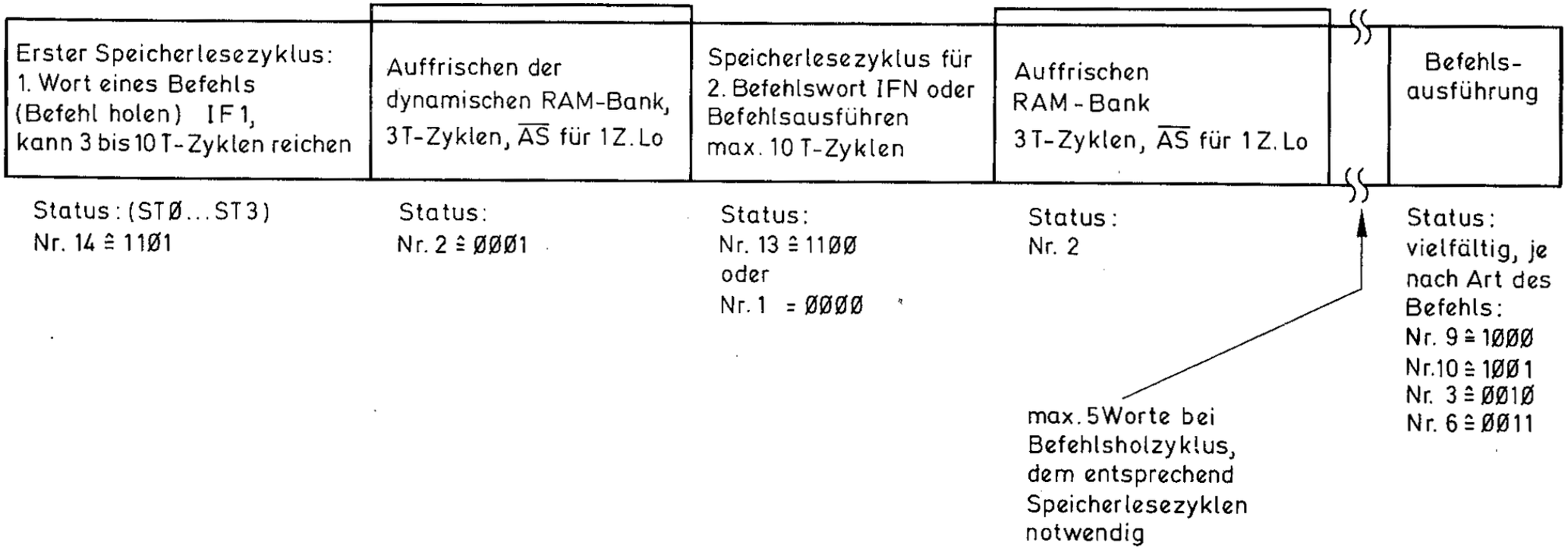


Bild 2.33

nungszyklus erfolgt. Es kann sich zwischen dem Befehlshol- und dem Anerkennungszyklus noch ein Auffrischzyklus hineinschieben. In dem Befelsholzyklus wird jedoch intern ein Prüfverfahren eingeleitet, das einen Interrupt anerkennt oder nicht. Dies trifft jedoch mehr für einen VI oder NVI zu, denn es gibt praktisch keinen Grund für die CPU, einen NMI nicht anzuerkennen, es sei denn, daß noch vor einem NMI ein interner Trap aufgetreten ist, der dann zuerst behandelt wird. Dieser Fall wäre aber mehr theoretischer Natur. Ein NMI wird als sehr wichtiges Ereignis angesehen. Ein NMI muß ausgeführt werden, weil das NMI-Flag nicht zurückgesetzt werden kann. Der Anerkennungszyklus besteht aus 10 CPU-T-Zyklen, wobei nach dem zweiten automatisch 5 Wartezyklen eingefügt werden. $\overline{\text{WAIT}}$ muß jedoch, wie im Bild gezeigt, gesetzt werden. Im ersten T-Zyklus erscheinen nun endlich auch der entsprechende Anerkennungs-Status. Ein VI oder NVI muß so lange am CPU-Eingang angelegt bleiben, bis ein Anerkennungs-Status erschienen ist. Der Anerkennungszyklus hat eine eindeutige Aufgabe. Von dem peripheren System, das die Unterbrechung ausgelöst hat, wird eine Kennung abgefragt, die mit dem folgenden Rettvorgang in den System-Stapelspeicher geschoben wird. Bei einem Trap war diese Kennung bereits am Bus. Es war das erste Befehlswort eines Befehls, das diesen Trap auslöste. Bei einem VI ist dieses Wort sehr wichtig, denn aus dem unteren Byte wird der Sprungvektor abgeleitet, der zusammen mit der vom PSAP ausgegebenen Adresse den Startpunkt für die Interrupt-Serviceroutine eines VI liefert. Bei einem MMU-Segment-Trap ist beispielsweise das obere Wort der Kennung wichtig. Der Anerkennungszyklus zeigt im ersten CPU-T-Zyklus ein aktiviertes $\overline{\text{AS}}$, was auch eine Adresse zur Folge hat. Diese sollte im Anerkennungszyklus nicht beachtet werden. In **Bild 2.39** wird auf das Kennungs-Wort noch einmal hingewiesen. In **Bild 2.40** wird nochmal eindeutig gezeigt, was nach dem Anerkennungszyklus in den Systemstapel-Speicher gerettet wird. Nach einem Anerkennungszyklus ist die CPU automatisch in den System-Modus gesetzt worden, der Z 8001 überdies noch in den segmentierten Modus. Der System-Stapelzeiger wird zuerst um zwei Stellen dekrementiert und sodann das erste Wort, der PC-Offsetwert, gerettet.

$\overline{\text{RESET}}$ ist bekanntlich eine Funktion, bei der jegliche Arbeitsprogrammtätigkeit der CPU schlagartig gestoppt wird. Ist der $\overline{\text{RESET}}$-Eingang aktiviert, wird generell der Programmzähler auf Null gesetzt (so beim 8085 und dem Z 80). Beim Z 80 wird ebenso das Interrupt-Flipflop zurückgesetzt, ferner das Index- und Refresh-Register auf Null gesetzt. Eine weitere Initialisierung wird nicht eingeleitet. Eine gewisse Ausnahme bildet die 6502 CPU. 6 Taktzyklen nach $\overline{\text{RESET}}$ wird das Maskeninterruptflag gesetzt und der Programmzähler mit den Vektoren FFFCH und FFFDH geladen, um einen definierten Monitorstart zu sichern. Der Z 8000 startet ganz unten. Im Gegensatz zum Z 80 startet der Z 8000 mit einer echten Initialisierung. **Bild 2.41** zeigt das Antwortdiagramm auf eine RESET-Aktivierung. RESET muß mindestens 180 ns auf Lo gehalten werden, damit die CPU darauf reagieren kann. Wird RESET geöffnet, haben für die nächsten fünf Taktzyklen (oder auch mehr) der AD-Bus einen hochohmigen Zustand, sind die Signale $\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{MREQ}}$, $\overline{\text{BUSAK}}$ und $\overline{\mu\text{O}}$ im inaktiven Zustand (Hi), der Status und die Segmentleitungen Null, der Refresh gesperrt und die Signale R/$\overline{\text{W}}$, B/$\overline{\text{W}}$ und N/$\overline{\text{S}}$ in einem nicht definierten Zustand. Nach dieser Art Quarantänezeit von einer bis zwei Mikrosekunden beginnt die CPU etwas zu unternehmen. Es erscheint die Adresse 0002H am Bus. Diese Adresse ist dafür gedacht, die CPU wieder auf einen definierten Betriebszustand zu bringen. Die CPU holt sich das Statuswort für das FCW-Register. Nach zweifachem Inkrementieren (auf 0004H) holt sich die CPU die Segmentadresse

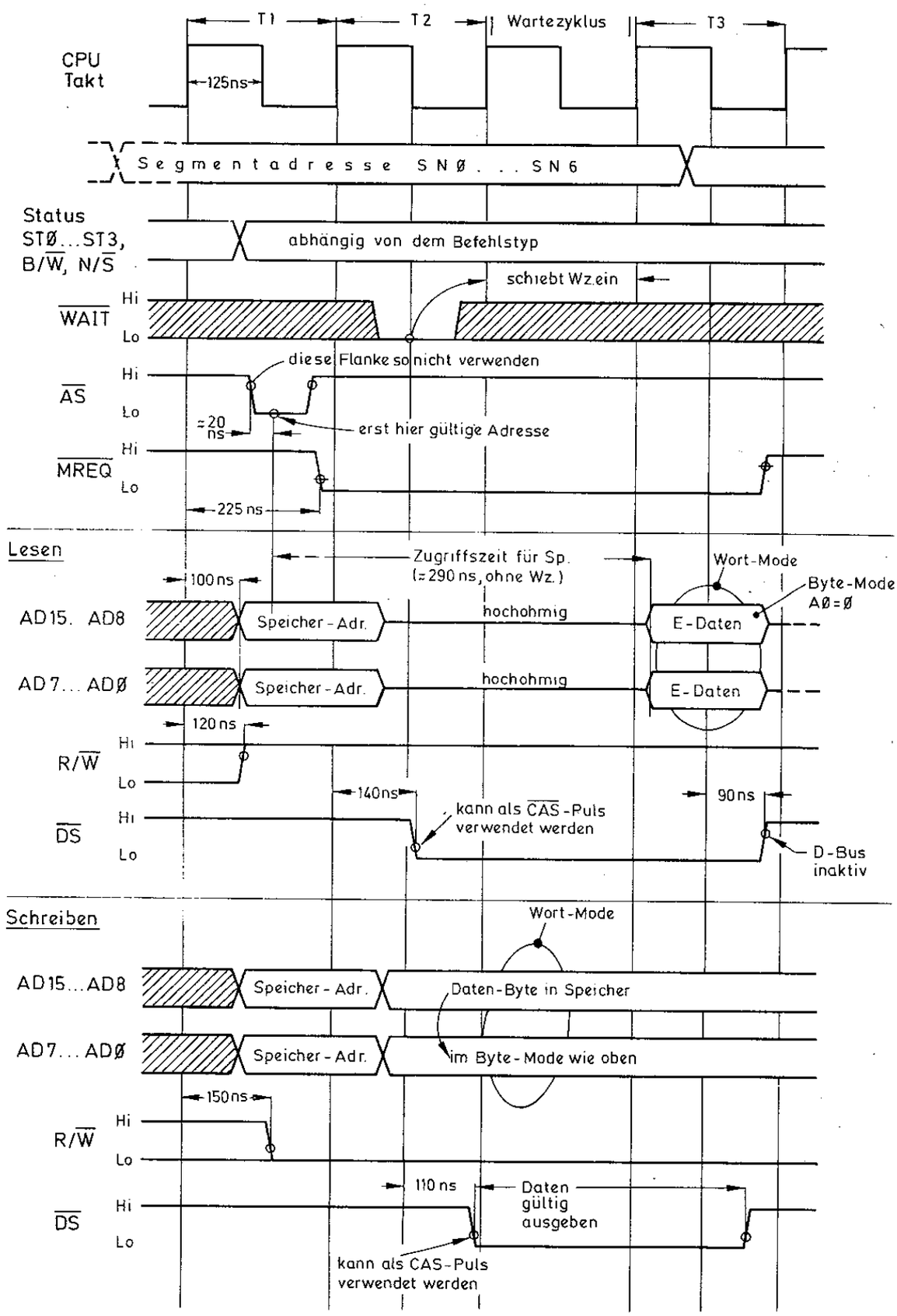

T1
T2
Wartezyklus
T3
CPU Takt
125ns
Segmentadresse SNØ . . . SN6
Status STØ...ST3, B/W, N/S
abhängig von dem Befehlstyp
schiebt Wz.ein
WAIT
Hi
Lo
diese Flanke so nicht verwenden
AS
≈20 ns
erst hier gültige Adresse
MREQ
225 ns
Lesen
Zugriffszeit für Sp. (=290 ns, ohne Wz.)
Wort-Mode
Byte-Mode AØ=Ø
100 ns
AD15. AD8
Speicher-Adr.
hochohmig
E-Daten
AD7... ADØ
120 ns
R/W
140ns
kann als CAS-Puls verwendet werden
90ns
DS
D-Bus inaktiv
Schreiben
Wort-Mode
Daten-Byte in Speicher
im Byte-Mode wie oben
150ns
110 ns
Daten gültig ausgeben
kann als CAS-Puls verwendet werden

Bild 2.34

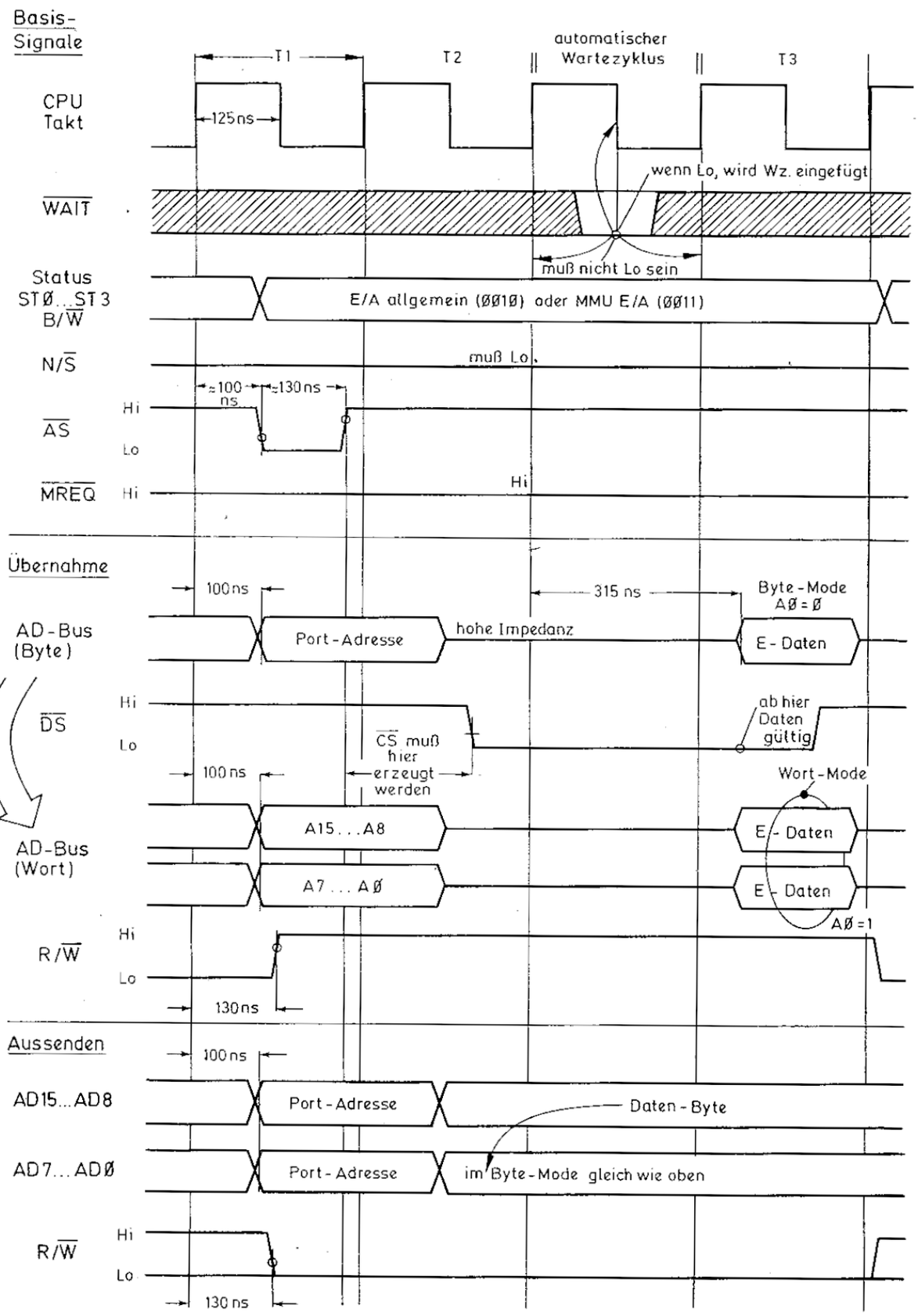

Basis-
Signale
T1
T2
automatischer
Wartezyklus
T3
CPU
Takt
125ns
WAIT
wenn Lo, wird Wz. eingefügt
muß nicht Lo sein
Status
ST0...ST3
B/W
E/A allgemein (00 10) oder MMU E/A (00 11)
N/S
muß Lo
≈100 ns
≈130 ns
AS
Hi
Lo
MREQ
Hi
Hi
Übernahme
100ns
315 ns
Byte-Mode
A0 = 0
AD-Bus
(Byte)
Port-Adresse
hohe Impedanz
E-Daten
DS
Hi
Lo
ab hier
Daten
gültig
CS muß
hier
erzeugt
werden
100 ns
Wort-Mode
A15...A8
E-Daten
AD-Bus
(Wort)
A7...A0
E-Daten
A0 = 1
R/W
Hi
Lo
130ns
Aussenden
100ns
AD15...AD8
Port-Adresse
Daten-Byte
AD7...AD0
Port-Adresse
im Byte-Mode gleich wie oben
R/W
Hi
Lo
130 ns

Bild 2.35

(nur Z 8001) und schließlich bei der Adresse 0006H holt sie sich den Adreß-Offset. Zeichnen sich diese drei EX-Zyklen durch die Statusausgabe Nr. 12 aus, beginnt die CPU, wenn man so will, bei der Offset-Adresse 0006H mit der Tagesordnung. Bei deiser Adresse wird der erste Holvorgang eingeleitet und entsprechend mit dem Status Nr. 13 abgesegnet. Für den Z 8002 sei hier nur bemerkt, daß bereits bei der Adresss 0004H der Offset geladen wird. An dieser Adresse könnte man zum Sprung in das Betriebssystem ansetzen. Der Sprung ab 0006H bzw. 0004H muß jedoch erst vorbereitet werden. An diesen Adressen muß ein Ladevorgang einer absoluten Adresse in ein Allzweckregister erfolgen. Dann erst kann mit einem Sprungbefehl JP diese Adresse in den Programmzähler geladen werden, die dann auf den Anfang des Betriebssystems hindeutet.

Man sollte einen weiteren Punkt nicht vergessen. Betreibt man in dem System dynamische RAMs, wird es sicher empfehlenswert sein, mit dem Befehl LDCTL REFRESH, R (X) erst einmal die Speicher wieder in Funktion zu setzen. Wird beispielsweise das Initialisierungswort für das Refresh-Register aus dem Register R 0 geladen, lautet das Hex-Format des Befehls dafür 7D0BH. Wird nun 9E00H in dieses Register geladen, ist ein Auffrischzyklus von 15 μs gesichert.

Bild 2.42 zeigt das Zeitdiagramm der CPU-Signale bei einem HALT-Befehl. Die CPU arbeitet bei diesem Befehl eine unbegrenzte Zahl interner Zyklen ab. Der Programmzähler wurde um 2 Stellen erhöht. Wird PC + 2 ausgegeben, so wird an den Statusleitungen ein interner Zyklus ausgegeben (Status Nr. 0). Je nach Programmierung wird ein Auffrischzyklus eingeschoben. Bei diesem Befehl wartet die CPU sozusagen auf einen Interrupt oder Segment-Trap, der sie aus dem HALT befreit. Ein Segment-Trap kann es niemals sein, weil die CPU keinen unerlaubten Zugriff macht. Im HALT kann sie nichts anstellen. Es kann in der Tat nur ein Interrupt sein. Es wird jedoch auch der $\overline{\text{SEGT}}$-Eingang mit abgefragt, weil er in die Abfrageroutine mit inbegriffen ist. Die Abfrage erfolgt immer zwischen dem zweiten und dritten Takt eines internen oder Auffrisch-Zyklus. Die Abfrage der Unterbrechungseingänge erfolgt immer zwischen diesen beiden Takten. Die Abfrage geschieht zweimal, wenn das Ergebnis in einen Refreshzyklus gefallen ist, d.h. es wird im nächsten internen Zyklus noch einmal abgefragt. Ein NMI macht hier keine Ausnahme. Wenn das NMI-Flag durch ein Ereignis gesetzt wurde, erfolgt eine Anerkennung ohnehin. Wurde ein Ereigniseingang als aktiv anerkannt, folgt ein schon im Bild 2.38 festzustellender IF 1-Zyklus. Auch hier wird wiederum entschieden, bei welchem Ereignis die höchste Priorität liegt, wenn beispielsweise mehrere Interrupts aufgetreten sind. Erst nach diesem IF 1 folgt der eigentliche Anerkennungszyklus. Ausgenommen RESET sind nur die genannten Ereignisse in der Lage, die CPU aus einem HALT-Zyklus zu befreien. Wurde die Serviceroutine ausgeführt, wird die CPU wieder in den HALT-Zyklus zurück-

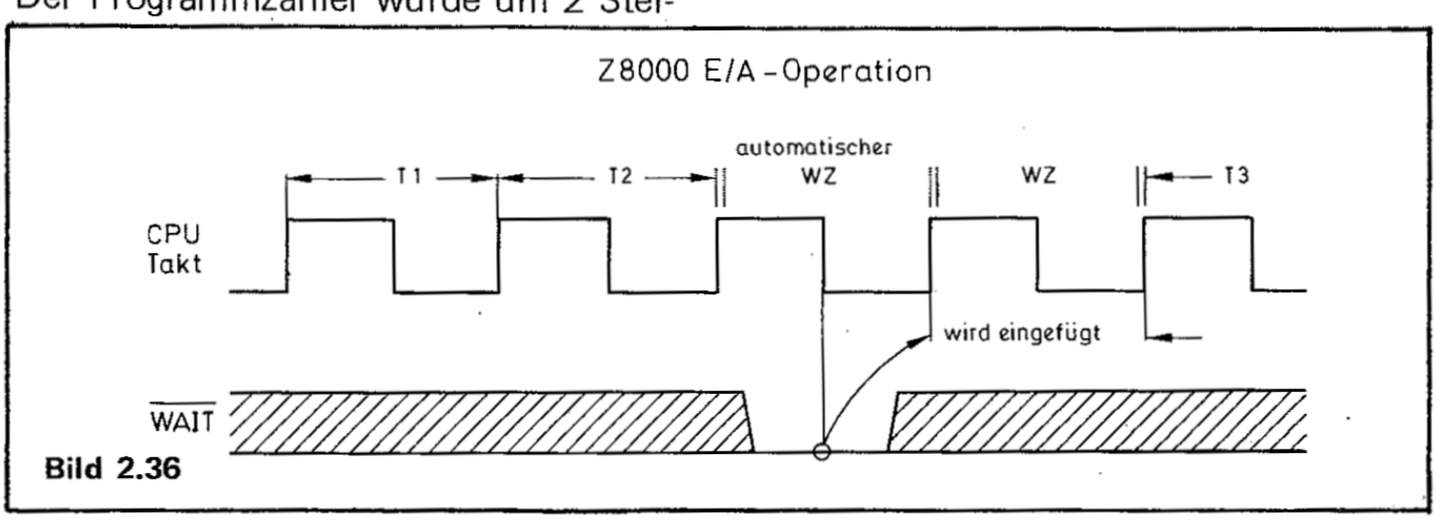


Bild 2.36

kehren. Das muß auch so sein, denn nach den Regeln einer Programmunterbrechung kehrt die CPU wieder dahin zurück, woran sie vor der Unterbrechung gearbeitet hat.

Im Gegensatz zum NOP-Befehl ist HALT ein priviligierter Befehl, obwohl die Funktion beider die gleiche ist. Auch bei NOP wird der PC um 2 Stellen inkrementiert.

Der $\overline{\text{STOP}}$-Eingang der CPU (beim Z 8001 Pin 7, beim Z 8002 Pin 6) wird zur Einzelschrittsteuerung benutzt. Einzelschrittsteuerung bedeutet, daß man einen Befehl quasi isoliert ausführen kann. **Bild 2.43** zeigt das Signaldiagramm nach Aktivieren des $\overline{\text{STOP}}$-Eingangs. Der $\overline{\text{STOP}}$-Eingang wird während der Hi ↓ Lo-Flanke des letzten Taktes irgendeiner Befehlsausführung abgefragt. War dieser Lo, holt die CPU nur noch das erste Wort des nächsten Befehls (Status Nr. 13) und beginnt nach den drei T-Zyklen mit einem Refreshzyklus. Der $\overline{\text{STOP}}$-Eingang wird wieder im letzten Takt, hier T 3, abgefragt und bei Lo ein weiterer Refresh-Zyklus angefügt. Ist bei der nächsten Abfrage STOP weiterhin Lo, folgt ein weiterer RZ. Dies setzt sich unbegrenzt fort. Im Gegensatz zum programmierten Ablauf im Refresh-Register folgen ständig RZs, wenn in der Mitte des dritten Taktes $\overline{\text{STOP}}$ gleich Lo war. Hat die CPU zu irgendeinem dritten Takt eines RZ ein Hi festgestellt, fügt sie noch einen RZ hinzu und schließt den IF 1 mit den Takten T4 und T5 ab. Ein RZ wird von der äußeren Hardware einem internen Zyklus sehr ähnlich gesehen. Der Unterschied zwischen einem RZ und einem internen Zyklus besteht nur darin, daß die CPU bei einem RZ an dem $\overline{\text{MREQ}}$-Ausgang für zwei Takte Lo herausgibt. Beim Einzelschritt-Zyklus sind diese Haltzyklen jedoch effektive Auffrisch-Zyklen, weil $\overline{\text{MREQ}}$ bei jeder Adreßausgabe entsprechend Lo ist. Wenn $\overline{\text{STOP}}$ also Lo gesetzt wird, ist der vorhergehende Befehl ausgeführt worden. Nach dem Holen des ersten Wortes des nächsten Befehls wird die CPU auf unbegrenzte Zeit angehalten, solange $\overline{\text{STOP}}$ zu den genannten Zeitpunkten Lo zeigt. STOP ist somit taktsynchron.

Wie oft ein dynamischer Speicher gelesen oder aufgefrischt wird, ist im Prinzip gleichgültig. Man darf jedoch dabei nicht vergessen, daß bei einem ständigen Auffrischen der dynamischen Speicherbausteine (in 750 ns einmal) intern mehr Leistung verbraucht wird. Das äußert sich in jedem Fall in einer Erhöhung der Temperatur in den RAMs. Es könnte zu kritischen Werten kommen, wenn die Z 8000A-CPU längere Zeit bei $\overline{\text{STOP}}$ aktiv gehalten wird. Es ergibt sich dann praktisch alle 500 ns ein Refresh-Zyklus. Diese Häufigkeit kann die dynamischen RAMs schon kräftig "anheizen". Man muß jedoch ernstlich nach dem Sinn fragen, einen Z 8000A länger in dieser Art zu betreiben. Das wäre doch Unfug.

Es wäre von Vorteil, würde die CPU bei einem Einzelschritt-Zyklus so verfahren, wie es bei einem HALT-Befehl geschieht. Es liegt eben daran, daß $\overline{\text{MREQ}}$ stetig und zyklisch aktiv ist. Ausführungstechnisch ist es auf dem Z 8000-Chip sicher einfacher, $\overline{\text{MREQ}}$ ständig zu aktivieren. Andererseits dürfte es für einen versierten Hardwaretechniker keine zu große Schwierigkeit sein, die große Zahl an RZs zu reduzieren. Es ist auch in der Tat nicht allzu schwierig, eine Lösung hierfür zu finden. In **Bild 2.44** ist eine kleine Schaltung angegeben, die die Zahl der RZs zu reduzieren hilft. Ein 4-bit-Binärzähler 74LS93 wird an seinen höchstwertigen Q-Ausgängen auf die Rücksetz-Eingänge zurückgekoppelt. Dadurch kann der Zähler bis 12 zählen und wird danach wieder auf Null gesetzt. Die UND-Verknüpfung liefert nach dem zwölften $\overline{\text{MREQ}}$-Signal in dem kurzen Übergangszeitraum zweier RZs ein Hi. Gibt man diese Flanke in das Monoflop 74LS123, erhält man am $\overline{\text{Q}}$-Ausgang einen 800 ns Lo-Zyklus, der nach 12 $\overline{\text{MREQ}}$-Signalen einmal auftritt. Somit wird ein Auffrischzyklus erst nach 9 μs für die dynamischen RAMs

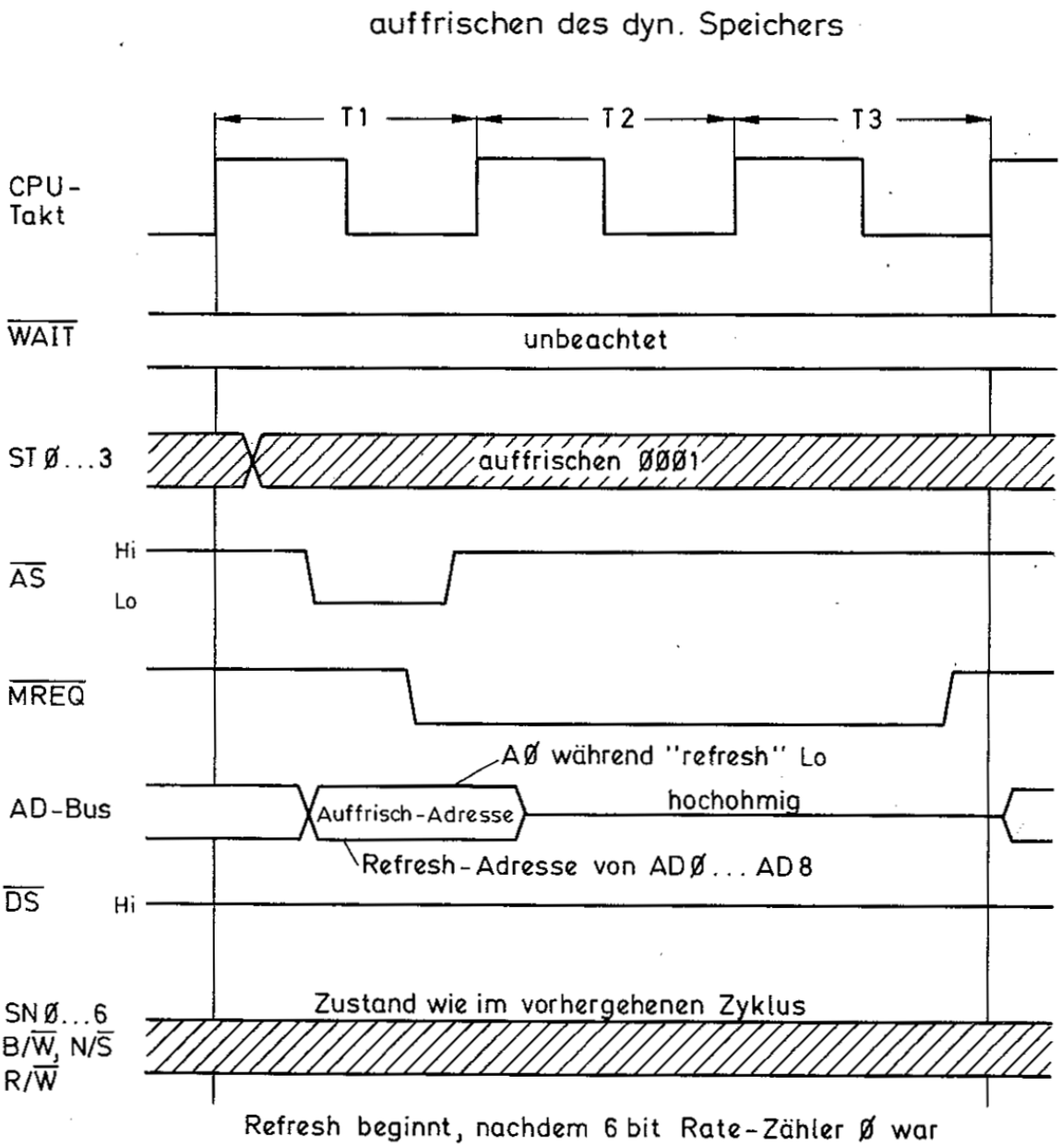


Bild 2.37

wirksam, weil erst dann das MREQ-Freigabesignal die Speicherbausteine erreicht. Die Verknüpfung über Tri-State-Gatter ermöglicht es, die Schaltung nur bei STOP aktiv wirksam zu machen. Die Einschaltverzögerung von STOP bewirkt, daß der erste Befehlsholzyklus nach aktiviertem STOP noch ausgeführt wird, bevor an den Tri-States auf die reduzierte MREQ-Ausgabe umgeschaltet wird. Bei einem Refreshtakt von 9 µs werden die dynamischen RAM-Bausteine sicher nicht zu hoch belastet.

Wie in Bild 2.42 zu sehen ist, tritt bei dem HALT-Befehl der Status Nr. 0 (interne Operation) auf. Die CPU steht eben nie ganz still, sofern Spannung und Takt anliegen. Interne Operationen werden auch bei anderen CPU-Befehlen angezeigt, wie z.B. bei der Multi-Mikro-Anforderung und bei den sehr zeitintensiven Operationen Multiplikation und Division und vereinzelt bei langen Verschiebeoperationen. Bei einer internen Operation wird nur eine Adresse ausgegeben keine Daten geholt oder ausgegeben. Im

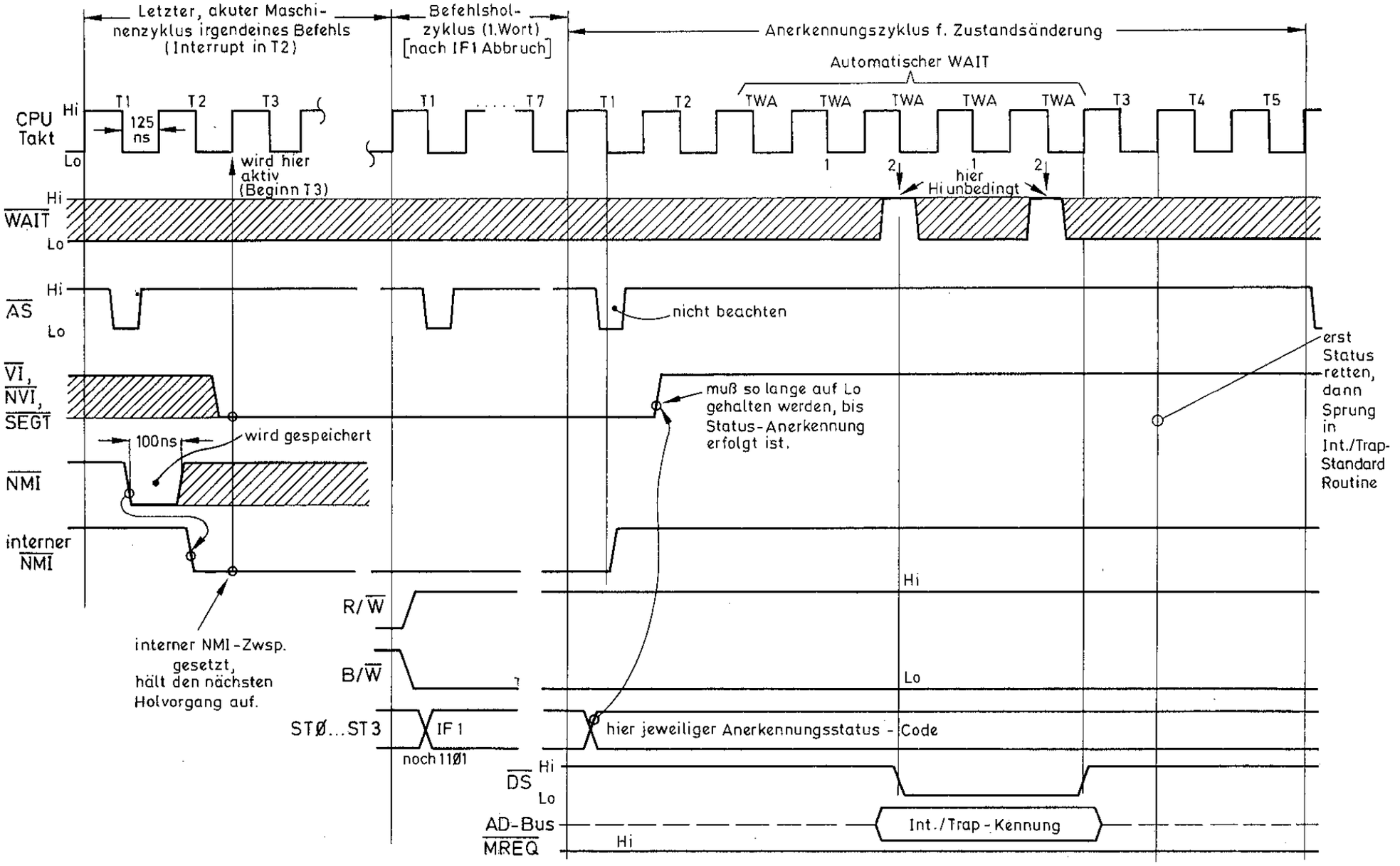


Bild 2.38

Zusammenfassend ist bei den in diesem Abschnitt erklärten CPU-Operationen bezüglich Hardware auffallend, daß die CPU in Dreier-Taktschritten arbeitet. Ohne Wartezyklus benötigt sie 750 ns bzw. 500 ns bei der A-Version.

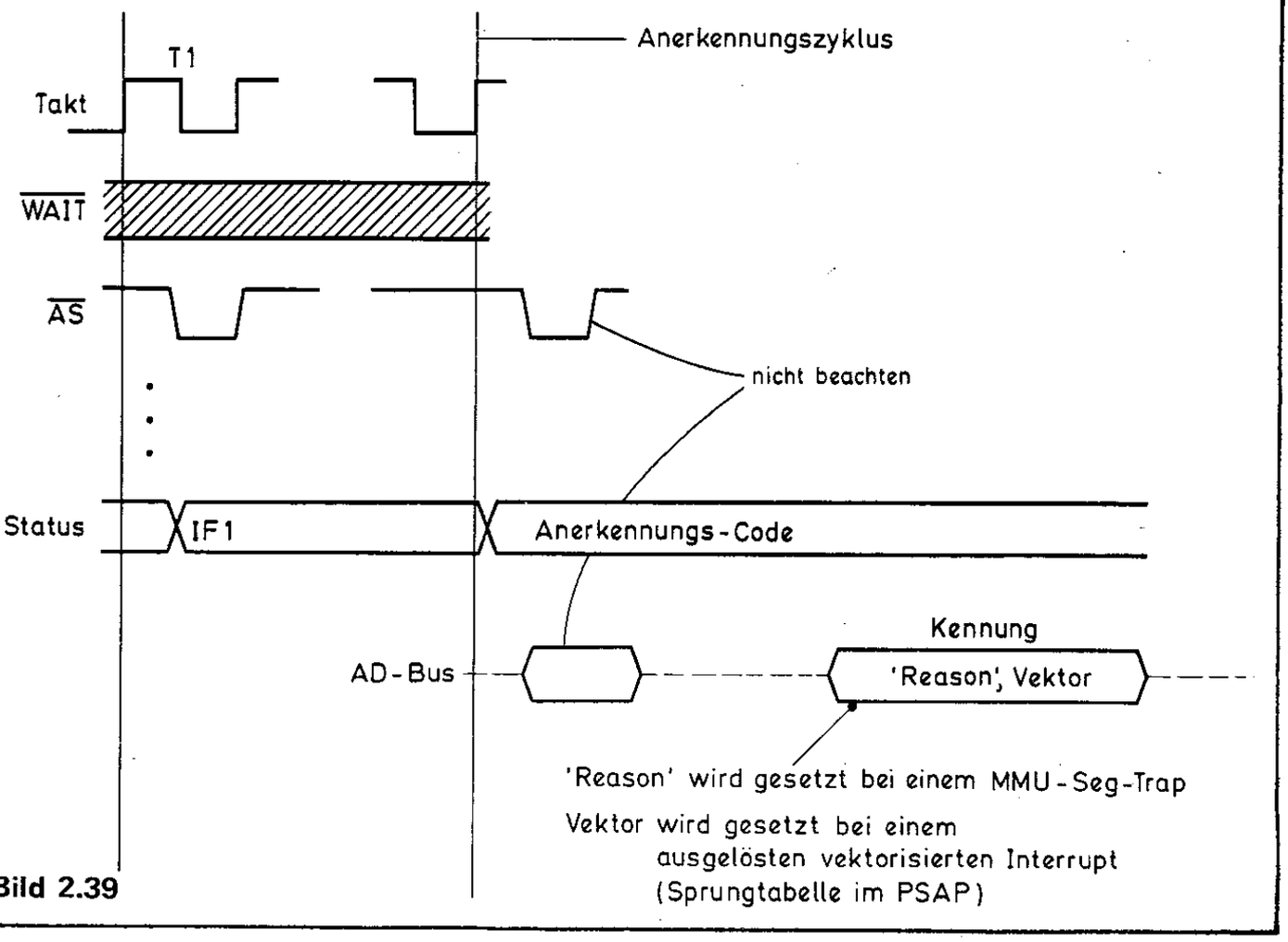


Bild 2.39

Gegensatz zu einem Refresh-Zyklus ist $\overline{\text{MREQ}}$ immer Hi. Refresh- und interne Operationen unterscheiden sich nur um eine Stelle im Status. Somit bringt die Darstellung eines internen CPU-Zyklusses in **Bild 2.45** nicht mehr viel neues.

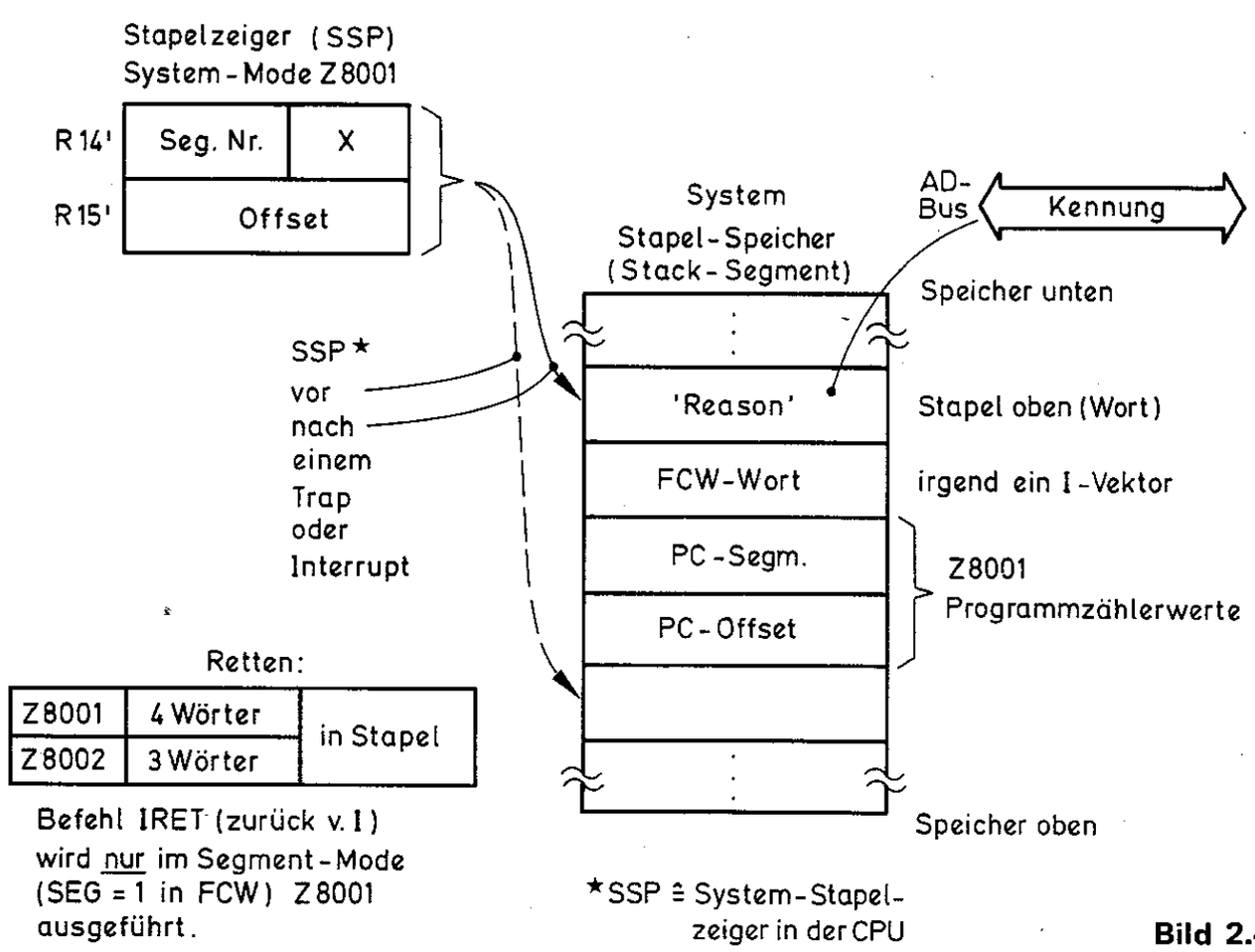


Bild 2.40

Bild 2.41

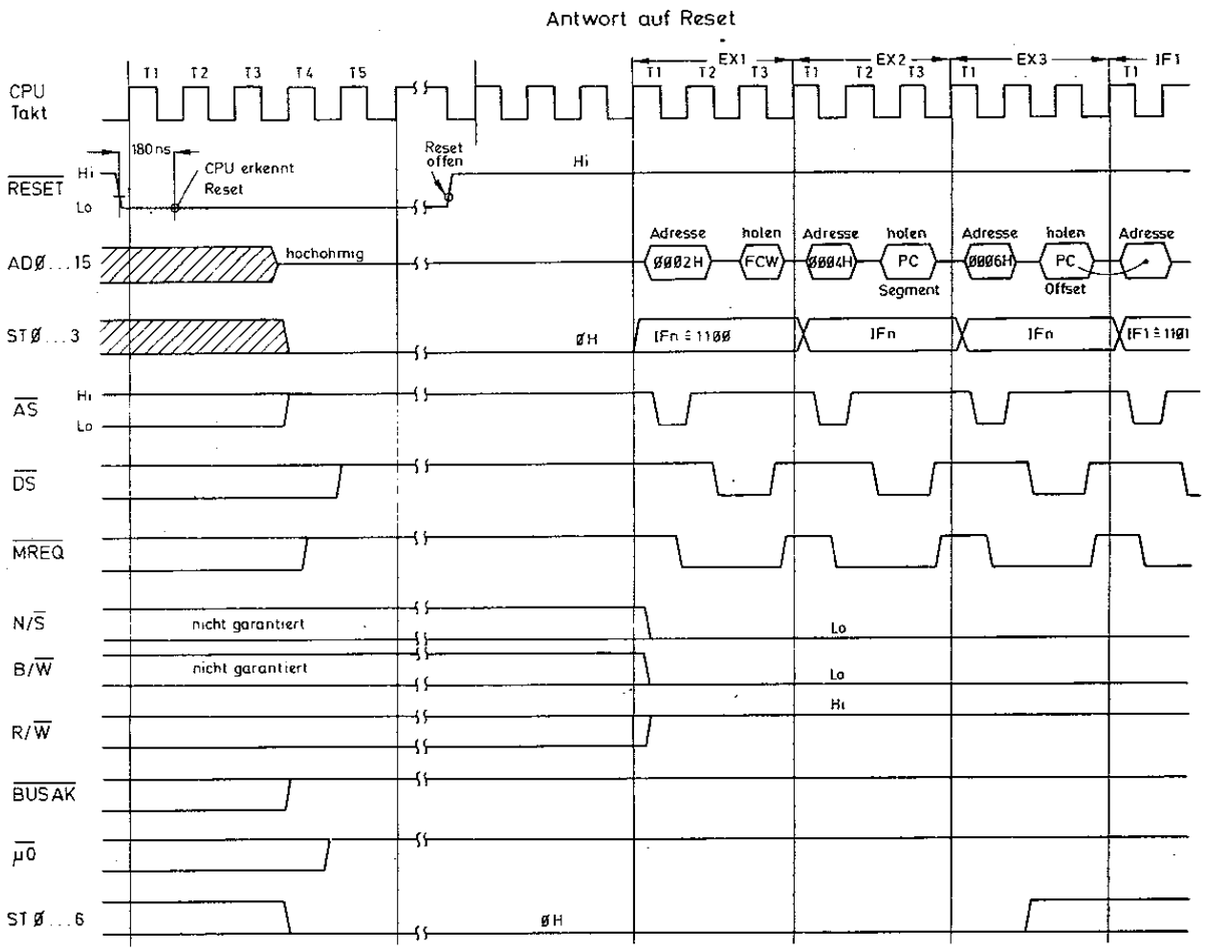


Bild 2.42

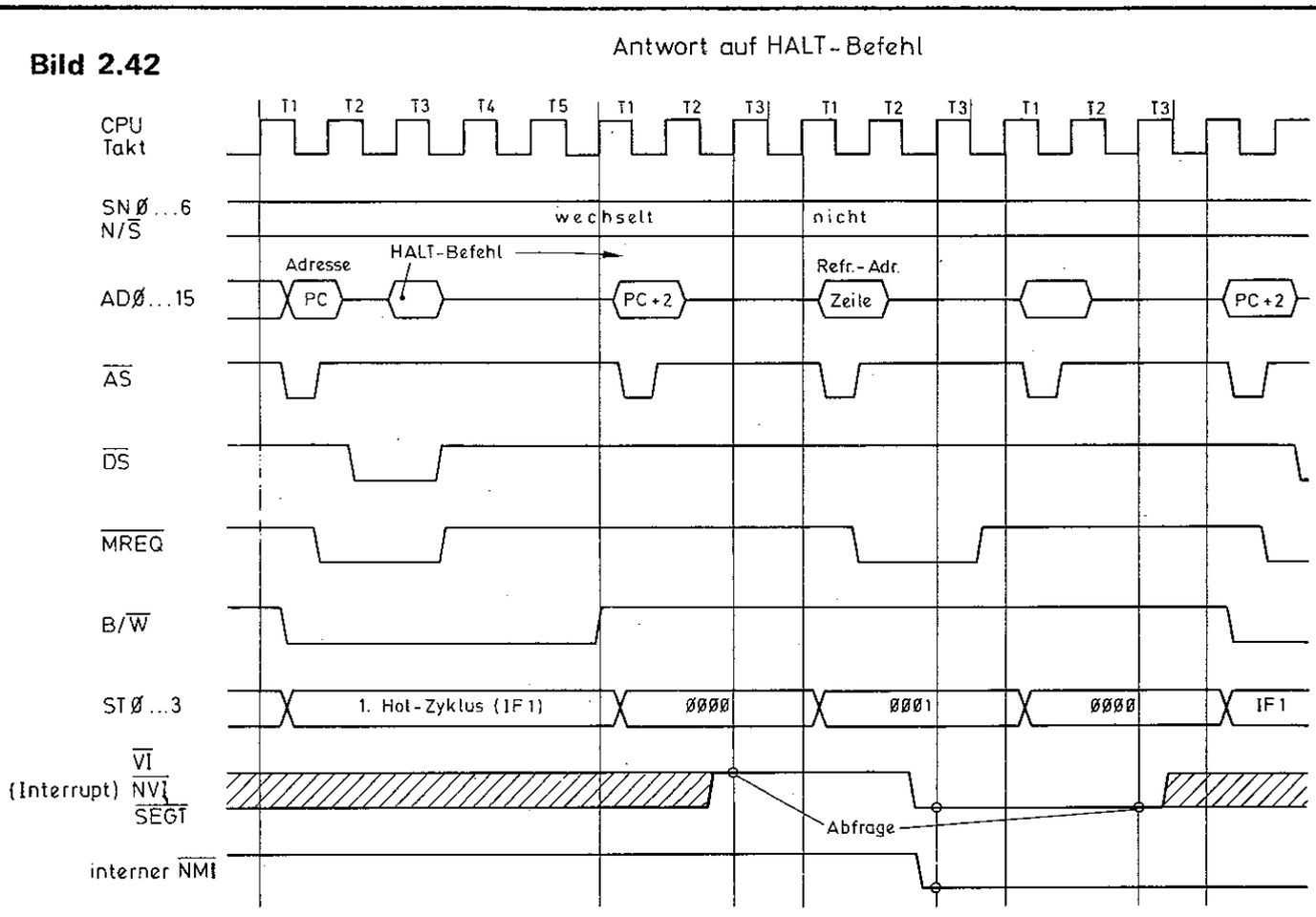


Bild 2.43

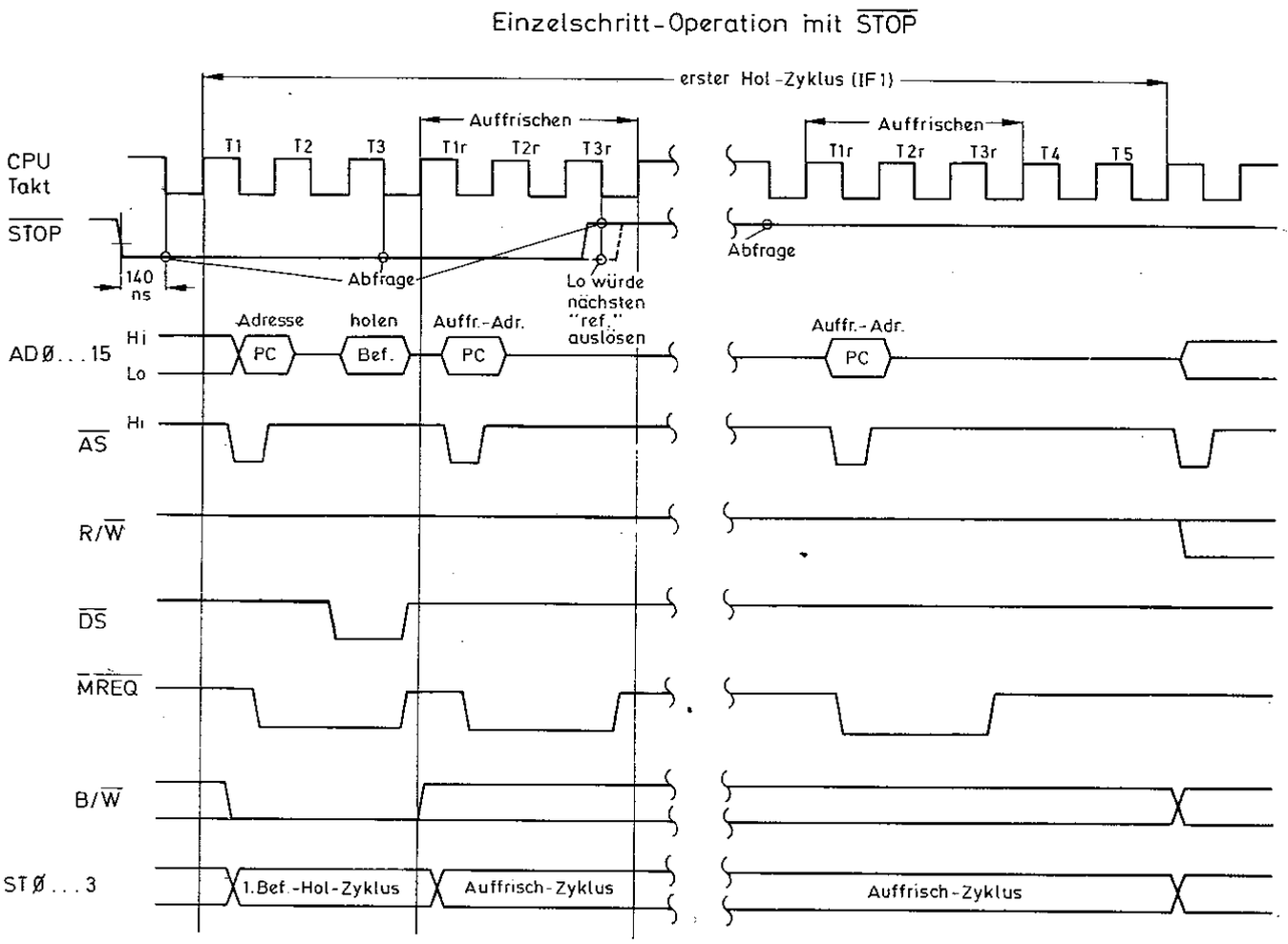
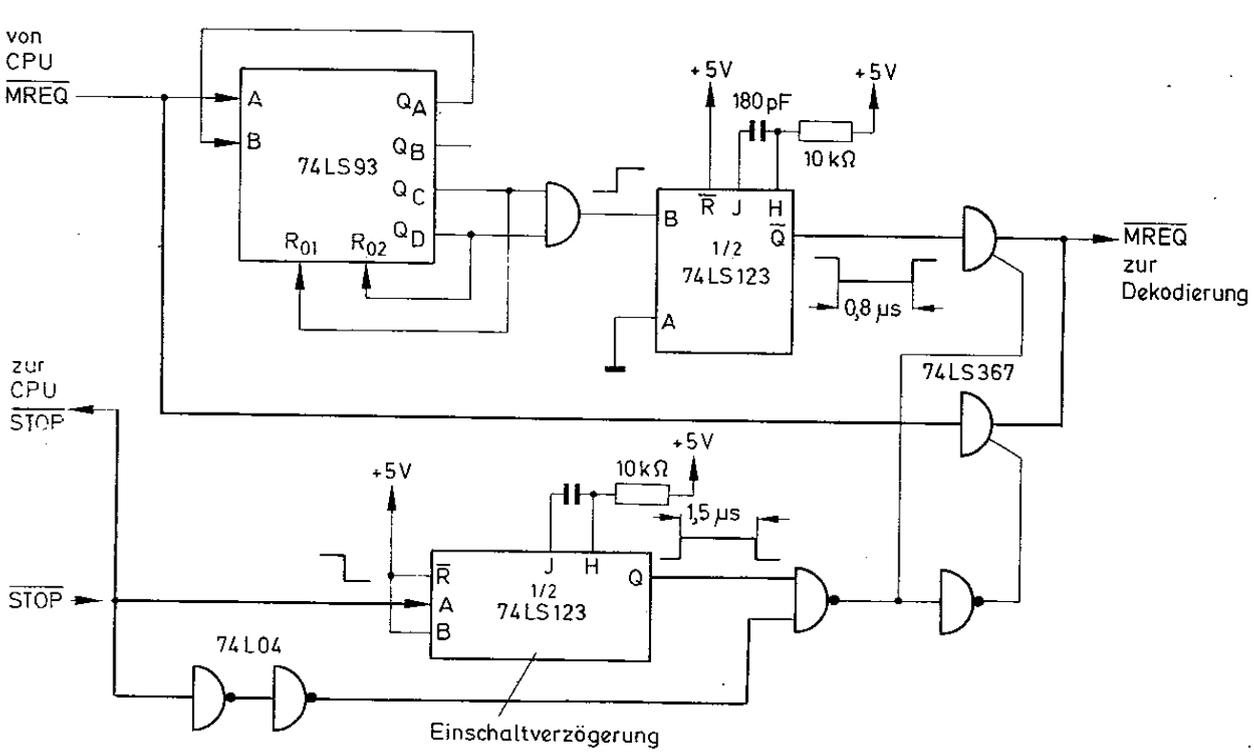


Bild 2.44



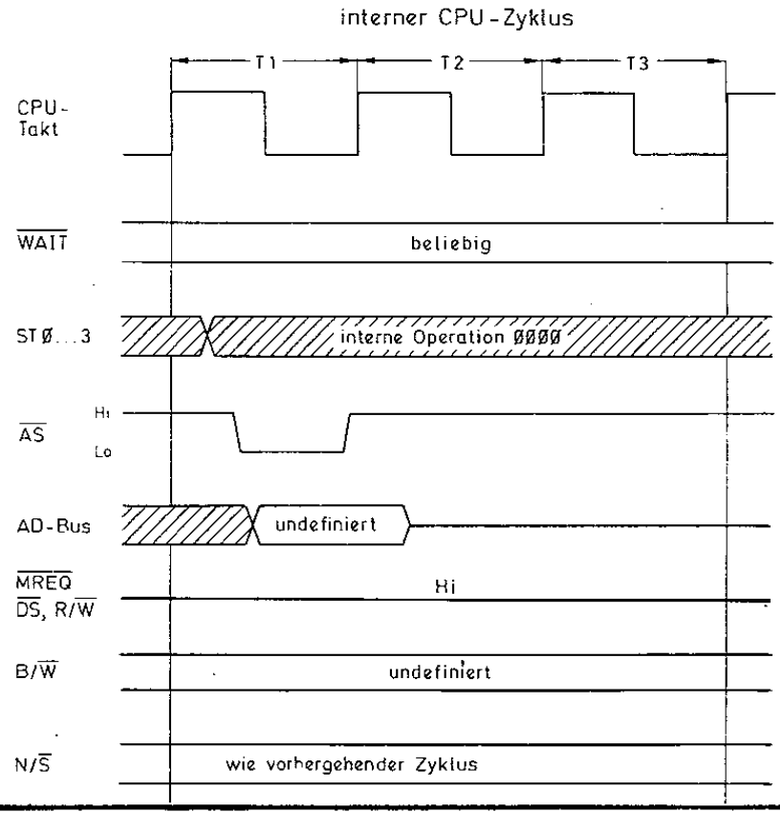


Bild 2.45

2.9 INTERRUPTBEHANDLUNG

Dem aufmerksamen Leser dieses Buches wird schon bis zu diesem Abschnitt der Begriff "Interrupt" als relativ häufig vorkommend aufgefallen sein. Interrupt ist in der Mikrocomputertechnik generell als ein häufig anzutreffender Begriff zu bezeichnen. Interrupts (Programmunterbrechungen) treten dann und zuweilen sehr häufig auf, wenn ein Computer eine größere Zahl von Aufgaben zugeteilt bekommt, die eine teilweise gleichzeitige, ineinander verschachtelte Bearbeitung erfordern. Diese Bearbeitung ist an bestimmte Bedingungen geknüpft, die einen synchronen Aufruf eines anderen Programms auf Grund von Ergebnissen im vorhergehenden Programm nicht einschließen, d.h. mit den Befehlen JP oder CALL ist das nicht zu machen. Der Sprung in ein anderes Programm wird ausschließlich von unvorhersehbaren Ereignissen geleitet, die unterschiedliche Natur besitzen. Der Prozessor kann nicht in Ruhe eine Aufgabe vor sich hin bearbeiten, sondern muß mehr oder weniger häufig eintreffende Anforderungen externer Ereignisquellen programmtechnisch behandeln. In der folgenden Beschreibung wird ein exemplarisches Beispiel gegeben.

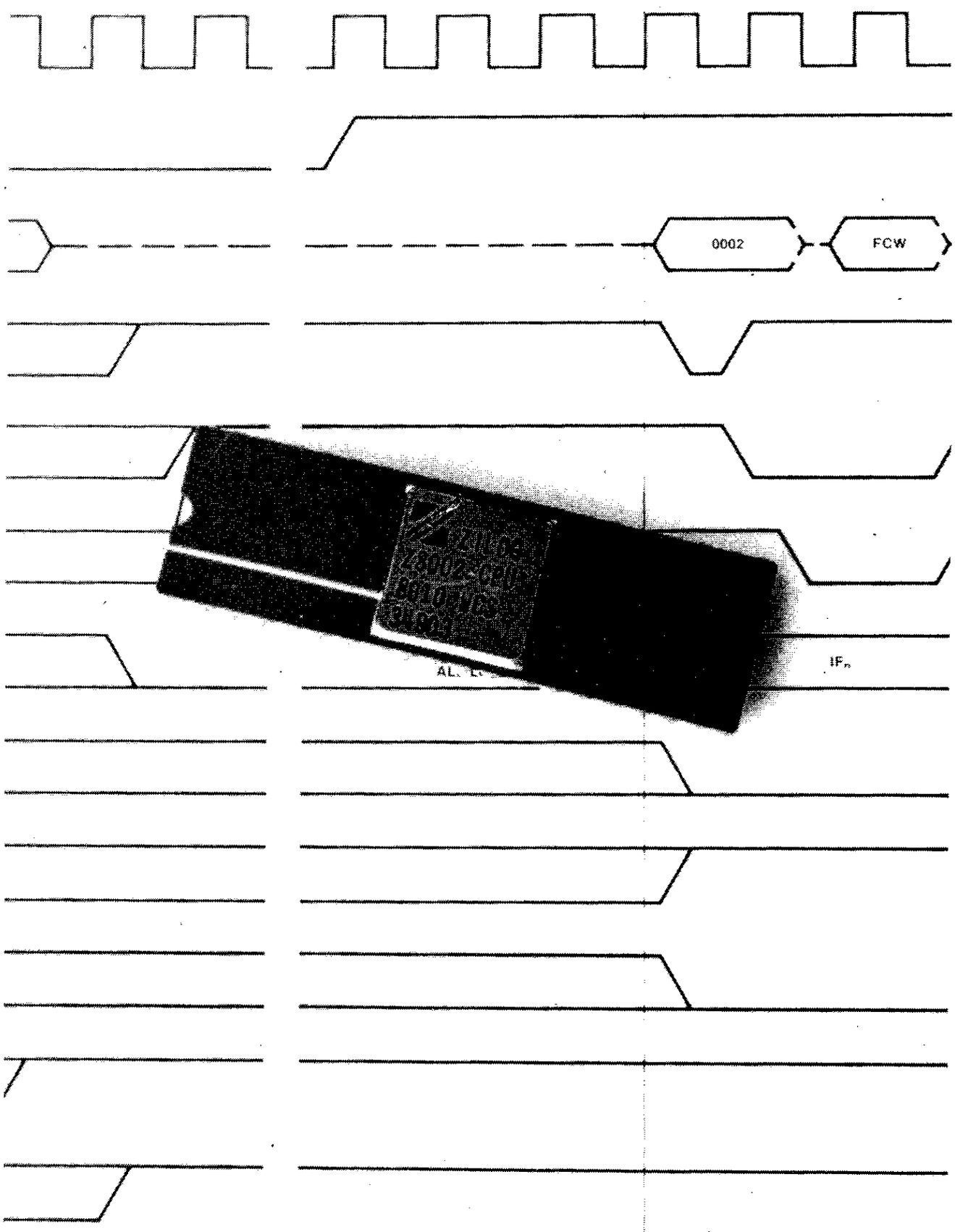
Bei einer gering automatisierten Warte einer technischen Produktionsstätte wird der mit der Aufsicht betreute Leiter eine Reihe von Prozeßvorgängen zu überwachen haben. Die Beobachtung der einzelnen Vorgänge (z.B. das Ablesen und Vergleichen von Meßwerten) des technischen Prozesses durch diesen Wartungsmann betrifft den kybernetischen Teil seiner Aufgabe. Dieses Modell muß für die weitere Betrachtung nochmals vereinfacht werden. Den einzelnen Meßwerten ordnet man eine Toleranz zu, die beim Austritt aus dem Toleranzbereich ein akustisches oder optisches Signal verursachen. Ist ein solches Signal aufgetreten, entspricht dies einer Anforderung auf eine Behandlung durch den Wartungsmann. Das Signal ist das auslösende Moment der Anzeige einer nicht mehr legalen Veränderung in einem Prozeßabschnitt. Das Signal deutet dem Wartungsmann an: "Hallo, komm her, im Prozeß ist was faul". Der Wartungsmann wird also hingehen, irgendwo routinemäßig drehen und damit einen Service erfüllen. Dieser Service genügt genau diesem Prozeßabschnitt und hat zum Ziel, ihn wieder in erlaubte Bereiche zu manövrieren. Treten mehrere

solcher Signale gleichzeitig auf, wird der Wartungsmann aus einer gewissen Erfahrung heraus zuerst **da** drehen, dann **dort** drehen und zuletzt **seitlich** drehen. Er macht das so, weil er weiß, **da** ist es wichtig, **dort** weniger wichtig und **seitlich** nicht besonders wichtig. Er bedient nach einer Priorität, die er auf Grund seines Wissens in seinem Gehirn fest formatiert hat. Es kann natürlich auch vorkommen, daß er "die Übersicht verliert" Der Einfachheit halber wird dieser Fall nicht weiter betrachtet. Dieses Beispiel hat einen praktischen Bezug. Man sollte in jedem Fall den regelungstechnischen Aspekt dieses Vorgangs außer acht lassen. Der Wartungsmann weiß, wie er drehen muß. Es geht vielmehr um den Aspekt der Bedienung und der Reihenfolge der Ereignisse. Die Behandlung der Ereignisse an sich soll einfach als Bearbeitungsroutine gesehen werden. Ferner ist noch wichtig, was der Wartungsmann tut, wenn er an keiner Stelle dreht. Er sitzt auf einem Stuhl und liest Zeitung. Physiologisch gesehen wären damit alle auftretenden Ereignisse angesprochen. Bringt man den Wartungsmann in ein besseres Licht und spricht von einem routinemäßigen Durchprüfen aller wichtigen Werte des Prozesses, so ist damit eine Tätigkeit im nicht aktiven Zustand angesprochen. Diese kontinuierliche Kontrolle ist die Arbeitsroutine, die er ausführt, wenn er nirgends dreht, d.h. an keiner Stelle eine Veränderung vornimmt. Letztlich wäre es gleichgültig, ob er Zeitung liest oder kontrolliert. Er ist auf jeden Fall bereit, einzugreifen, wenn er "gerufen" wird.

Dieses beschriebene Modell ist nun sehr gut auf die Mikroprozessorebene übertragbar. Die Z 8000-CPU simuliert gleichsam den Wartungsmann. Das Zeitunglesen entspricht dem Zustand des Prozessors, wenn er den HALT-Befehl ausführt. Die Routinekontrolle entspricht einem Programm einer Abfrage von Eingangsregistern aller Schnittstellen im System, die mit Kontrollwerten von außen geladen werden. Die akustischen oder optischen Signale entsprechen Strobe-Flags, die bei einem Ereignis (Austritt aus dem Toleranzbereich) gesetzt werden. Die Strobe-Flags sind alle in einer gewissen Hierarchie (geordnete Reihenfolge nach ihrer Bedeutung) an die CPU angeschlossen. Wird ein Strobe-Flag gesetzt, wird damit die Bearbeitung einer Service-Routine von der CPU angefordert. Die CPU wird in eine ganz bestimmte Service-Routine springen. In welche sie springen muß, wird sie entweder aus der Art des Interrupts ableiten können, das von dem Strobe-Flag angeregt wird, oder die anfordernde Schnittstelle selbst wird die Information liefern, welche Service-Routine die CPU bearbeiten muß. In der spezifischen Service-Routine (aus der Anregung folgend) ist gleichsam das Regelgesetz festgeschrieben, das den Prozeßabschnitt wieder ins rechte Lot bringt.

Erfolgt die Anforderung auf das Ausführen einer Service-Routine von mehreren Strobe-Flags fast gleichzeitig, entscheidet einerseits die Reihenfolge des Eintreffens und die Ebene in der Hierarchie, in der die Strobeleitungen zur CPU verdrahtet sind. Das liegt im Ermessen des Designers des Z 8000-Systems. Ist diese Verdrahtung festgelegt, ist damit auch die Reihenfolge der Bearbeitung oder gewisse Vorrechte einer Anforderung in das Schema der Z 8000-Interrupt-Hierarchie sozusagen fest maskiert. Kommen mehrere Anforderungen auf einen Service in kurzer Folge, wird die Z 8000-CPU als Wartungsmann niemals den Überblick verlieren, sondern in bürokratischer Manier nach "Schema F" diese Anforderungen bearbeiten. Wie eine Bearbeitung aus einer Anforderung eingeleitet wird, wurde bereits an einigen Bildern und Beschreibungen dargestellt, u.a. Anerkennungszyklus eines Interrupt in Bild 2.38, Rettvorgang des vorhergehenden Programmstatusses in Bild 2.40 und die Sprungtabelle des neuen Programmstatusbereichs einer Interrupt-Routine in Bild 2.22.

Wenn man noch einmal die Bedienung mehrerer Anforderungen betrachtet, wie sie der Wartungsmann ausführt, ist in


0002
FCW

seiner Handlungsweise des Drehens **da, dort** und **seitlich** eine weitere Beziehung hergestellt. Der Z 8000 unterscheidet mehrere Anforderungen auf drei Ebenen. Die CPU wird unmittelbar reagieren, wenn es gilt, **da** zu drehen. Eine Anforderung dieser Art entspricht der höchsten Ebene. Das ist die Ebene des **nicht maskierten Interrupts (NMI).** Wenn es gilt, **dort** zu drehen, ist der **vektorisierte Interrupt (VI)** angesprochen. Es muß auch irgendwann einmal **seitlich** gedreht werden. Dies sollte hier bedeuten, daß an dieser Stelle erst dann gedreht wird, wenn alle höherwertigen Drehvorgänge erledigt sind und nur noch dieser **seitliche** Bedarf übriggeblieben ist. Diese Ebene entspricht beim Z 8000 die eines **nicht vektorierten Interrupts (NVI).** Diese drei Möglichkeiten einer laufenden, asynchronen Unterbrechung einer Standardroutine (HALT, Abfrageprogramm oder beliebig) bietet der Z 8000. Die drei Ebenen liegen fest.

NMI → höchste Ebene
VI → mittlere Ebene
NVI → untere Ebene

Es gilt noch eine weitere Unterscheidung der drei Arten zu treffen. Zwischen dem nicht maskierten bzw. nicht vektorisierten Interrupt besteht ein entscheidender Unterschied. Ein vektorisierter Interrupt bestimmt über eine Zusatzinformation genau, welche Service-Routine bei einer Anforderung bearbeitet werden soll. In der Zusatzinformation liegt ein Zeiger bzw. Vektor fest, der genau auf die Startadresse der Service-Routine zeigt, die zur Erfüllung der Anforderung notwendig ist. Die Zusatzinformation wird mit dem $\overline{VI}$ von der Ereignisquelle geliefert. In Bild 2.39 wird auf dieses Kennungs-Wort im Anerkennungszyklus noch einmal hingewiesen. Das ist ein entscheidender systematischer Unterschied. Es ist nämlich möglich, auf einer Interruptebene mehrere Ereignisquellen priorisiert zu staffeln. Kommen nun mehrere NMI-Anforderungen, wird nach einer festverdrahteten Reihenfolge immer dieselbe NMI-Service-Routine von der CPU bearbeitet. Der Grund ist aus dem Bild 2.38 zu ersehen. Der Programmstatus-Bereichszeiger PSAP lädt bei einem NMI immer die gleiche Interrupt-Sprungadresse SSXX28H und somit immer die gleiche Service-Routine. Bei einem $\overline{NVI}$ ist das genauso. Hier lautet die Adresse nur SSXX30H. Bei einem vektorisierten Interrupt ist das anders. Jedes auslösende Interrupt-Ereignis hat, wenn man so will, eine Kennungsmarke. In dieser Marke wird der CPU in Form des genannten Zeigers mitgeteilt, welche Service-Routine das auslösende Element benötigt. Die Service-Routine kann somit auf die Bedürfnisse dieses Elements individuell zugeschnitten werden. VIs können aus diesem Grund eine hohe Bedeutung erlangen. Für alle VIs gilt ein typischer Wert des FCW, der an der Stelle SSXX38H des Programmstatus-Bereichsspeichers steht. Das VIE- und NVIE-Bit ist nicht gesetzt, so daß keine Unterbrechung dieser Interrupt-Serviceroutinen auftreten kann.

Wie entstehen nun diese Vektoradreßwerte? Der Programmstatus-Bereichsspeicher ist 256 Worte lang, wie bereits in Abschnitt 2.5 genannt wurde. Für den Z 8001 beginnt der Bereich des neuen Programmzählerwerts ab der Adresse SSXX3CH, wie aus Bild 2.22 zu ersehen ist. Der erste Vektor hat somit diesen Adreßwert. Das bedeutet, daß der Programmstatus-Bereichszeiger PSAP plus der Wert 3CH als Zielvektor die Startadresse für eine Service-Routine für das Element 0 bilden, das einen VI ausgelöst hat. Im Programmzähler befindet sich nach dem Retten des alten Programmstatus im Segmentteil SSH und im Offsetteil XX3CH. Das Kennungs-Wort für dieses Element (Vektor 0) muß im Anerkennungszyklus mm00H lauten. Der Wert mm ist das höherwertige Byte (beliebig) und 00H das niederwertige. Der nächste Vektor (Vektor 1) hat die Adresse SSXX40H oder PSAP + 40H für die entsprechende Service-Routine. Der Unterschied von 4 Worten ist leicht zu erklären. Sieht man sich noch einmal das Bild der Programmstatusregister in

Bild 2.20 an, fallen diese 4 Worte auf. Es wird auch für das nicht verwendete Register eine Stelle mitgerechnet.

Das Kennungs-Wort für den Vektor 1 lautet mm02H. Die folgende Tabelle zeigt einige Vektorwerte für die Startadressen von Service-Routinen.

Vektor 0	SSXX3CH	Kennung	mm00H
Vektor 1	SSXX40H	"	mm02H
Vektor 2	SSXX44H	"	mm04H
Vektor 3	SSXX48H	"	mm06H
Vektor 4	SSXX4CH	"	mm08H

usw.

Diese Vektoradressen sind nur für den Z 8001 zu setzen und wie man sieht, immer gerade, weil das Adreßbit A 0 gleich Null ist. Für den Z 8002 sind diese Werte etwas anders, weil bei einer Ladung des neuen Programmstatus nur 2 Wörter notwendig sind (FCW und PC-Offset). In Bild 2.22 sind praktisch alle Werte des niederwertigen Bytes zu halbieren. Die erste Vektoradresse beginnt also mit dem Wert XX1EH. Die nächste lautet XX20H, dann XX22H . . . Die folgende Tabelle zeigt die Vektorwerte für die Startadressen von Service-Routinen bei Verwendung des Z 8002.

Vektor 0	XX1EH	Kennung	mm00H
Vektor 1	XX20H	"	mm01H
Vektor 2	XX22H	"	mm02H
Vektor 3	XX24H	"	mm03H
Vektor 4	XX26H	"	mm04H

usw.

Durch die Möglichkeit, für den Statusbereichsspeicher des Z 8002 auch ungerade Werte einzusetzen, bietet diese CPU **256 Vektoren** Platz, wogegen der **Z 8001** nur **128 Vektoren** adressieren kann. In der Kennung des Z 8002 sind gegenüber des Z 8001 auch ungerade Werte enthalten.

Mit einem Sprung in die Interruptroutine ist zwar der Programmstatus des vorhergehenden Programms gerettet, jedoch nicht der Inhalt der Allzweck-Register. Wie vorhergehend beschrieben, wäre es bei der Ausführung von HALT- oder der Schnittstellenabfrage nicht notwendig zu retten. Wenn man sich noch einmal das Programmbeispiel in Abschnitt 2.5 ansieht (insbesondere Bild 2.16), hätte beispielsweise ein akzeptierter Interrupt nach der Zeile 6 die Folge, daß die Registerinhalte teilweise verloren gehen. Man muß hier Vorkehrungen treffen und retten, was es zu retten gibt. Wenn man retten muß, baut man nach dem Eintritt in die Interruptroutine unmittelbar einige PUSH-Befehle ein, die die noch unversehrten Registerinhalte in den Speicher schieben. Man nimmt dafür zweckmäßigerweise den System-Stapelzeiger und den entsprechenden Speicherbereich. Mit PUSHL-Befehlen können Doppelregister hinausgeschoben werden. Der Stapel verhält sich wie ein Stapel Dominosteine. Man kann nur einige oben drauf legen oder oben welche wegnehmen. Jeder Stein entspricht einem Wort. Neben dem PUSH-Befehl kann auch der Befehl LDM (mehrfaches Laden von Registern in den Speicher und zurück) angewandt werden. Hierzu ist ein beliebiger Speicherbereich verwendbar, jedoch **nicht der Stapelspeicher.** Man kann die Befehle PUSH und POP mit LDM simulieren, jedoch nicht für den Stapel einsetzen. Das Register mit dem Inhalt der Startadresse für den Transfer bei LDM wird inkrementiert und nicht dekrementiert. Hat die CPU die Interrupt-Serviceroutine fertig bearbeitet, werden mit entsprechenden POP-Befehlen die obersten Stapelworte wieder in die Z 8000-Register zurückgeholt. Daraufhin kann mit dem IRET-Befehl der alte Status des Programms vor dem Interrupt vom System-Stapel wieder geladen werden und alles ist in bester Ordnung.

Bei dieser letzten Beschreibung ist man auf den einzigen Nachteil gestoßen, den hochgradige Registerstrukturen wie des Z 8000 zeigen. Ist man gezwungen, Registerinhalte zu retten, muß man hier eben mehr Inhalte retten als bei anderen Prozessoren. Man kann jedoch darüber wirklich streiten, ob das ein Nachteil ist. Man muß doch nur die Inhalte retten, die wirklich wichtig sind. Außerdem sind mit drei PUSHL-Befehlen sechs 16-

bit-Registerinhalte zu retten. Wenn man die Befehle im segmentierten Modus ausführt, benötigt die CPU dafür 14,4 µs. So viel Zeit wird wohl bei der Interrupt-Serviceroutine noch zusätzlich übrig sein. Man hat in jedem Fall die 100%ige Gewähr dafür, nach Beendigung der Interruptroutine exakt an der gleichen Stelle des alten Programms weiterarbeiten zu können, an der zuvor abgebrochen wurde.

Die Hardware eines Interruptanschlusses verschiedener Ereignisquellen an die CPU sieht praktisch stereotyp aus. Es ist ein monotones Verdrahtungsschema für alle drei Arten von Interrupts. **Bild 2.46** zeigt das Schema dieser Verdrahtung, wie man es für eine Ebene anwendet. Es hat den Charakter einer Schleife. Eine Interrupt-Anforderung wurde vom System B angezeigt und auf der gemeinsamen Leitung aller Ereignisquellen an einen Interrupteingang übergeben. Die CPU zeigt über die Betriebszustandsanzeige am Ausgang der Dekodierung an, welchen Interrupttypus sie akzeptiert hat. Dementsprechend wird ein INTACK-Signal am Dekoder aktiv. Auf einer Interruptebene können mehrere Ereignisquellen priorisiert gestaffelt werden, wie Bild 2.46 zeigt. Dies geschieht nach einem "daisy-chain"-Busprinzip. Die Priorität einer Ereignisquelle liegt auf einer Ebene eindeutig fest: Das System A hat an seinem Priorisierungseingang IEI (Interrupt-Freigabe, aktiv Hi) immer grünes Licht, was gleichsam der höchsten Priorität in einer Ebene gleichkommt. Die Hardware-Priorisierung ist bei den Zilog-Bausteinen generell und bereits frühzeitig eingeführt worden. Diese Methode wird bei allen Z 80-Peripheriebausteinen wie der PIO (Parallel-Port), der SIO (Seriell-Port), des CTC (Zähler und Timer), des DMA (direkter Speicherzugriff) und des neueren Bausteins Z 80-DART (Zweifach Duplex Seriell-Empfänger/Sender) angewandt. Es ist selbstverständlich, daß das "daisy-chain"-Prinzip speziell bei den Schnittstellen-Bausteinen des Z 8000-Systems wie den Z 8030 SCC (Seriell-Schnittstelle) und den Z 8034 UPC (universelle Parallel-Schnittstelle) übernommen wurde. Bis zu 10 Bausteinen dieser Art können nach diesem Busprinzip verbunden sein, ohne daß ein zusätzlicher WAIT-Zyklus bei der Bedienung eingeschoben werden muß. Für den Speicherorganisationsbaustein Z 8010 MMU und den segmentierten DMA-Steuerbaustein Z 8016 DMATC gelten hier etwas andere Kriterien. Die MMU ist ein Kernbaustein und benötigt ein Businterface dieser Form nicht. Der Super DMA Z 8016 kann mehrere CPU-Funktionen übernehmen und verfügt über unterschiedliche Businterface-Formen, jedoch auch über diese "daisy-chain"-Formation.

Die vier Signalleitungen $\overline{\text{INTACK}}$ (Anerkennung), IEI (Freigabe-Eingang), IEO (Freigabe-Ausgang) und $\overline{\text{INT}}$ (Anforderung) ist die typische Busstruktur dieser Hardware-Interruptpriorisierung auf einer Ebene. Das Anforderungssignal $\overline{\text{INT}}$ ist ein offener Drain-Ausgang. Hat das System B (Bild 2.46) eine Aktivierung seines Strobe-Flags erfahren, wird $\overline{\text{INT}}$ nur dann Lo, wenn der Freigabe-Eingang IEI gleich Hi ist. Das ist hier der Fall. Deshalb wird System B bedient. Zugleich erzeugt es am Ausgang IEO ein Lo, was dem nächsten System anzeigt, daß eine Anforderung seinerseits (noch) nicht akzeptiert wird. $\overline{\text{INT}}$ des nächsten Systems kann nicht Lo werden. Erst wenn IEI von System B gleich Hi wird (Routine für System B beendet), kann das nächste System bedient werden. Von diesen Systemen führt nun der AD-Bus in voller Länge oder das untere Byte (AD 0 . . . AD 7) zur CPU, über den das einzelne, aktivierte System zu Beginn der Interrupt-Transaktion die Kennung ausgibt und später der Datentransfer erfolgt. Im Gegensatz zu den Z 80-Peripheriebausteinen besitzen die Z 8000-Schnittstellenbausteine noch den Anerkennungseingang $\overline{\text{INTACK}}$. Dieser Eingang ist gleichsam als Freigabe bzw. Anerkennung der entsprechenden Interruptebene zu verstehen, die die CPU intern priorisiert. Der nicht maskierte Interrupt muß immer ausge-

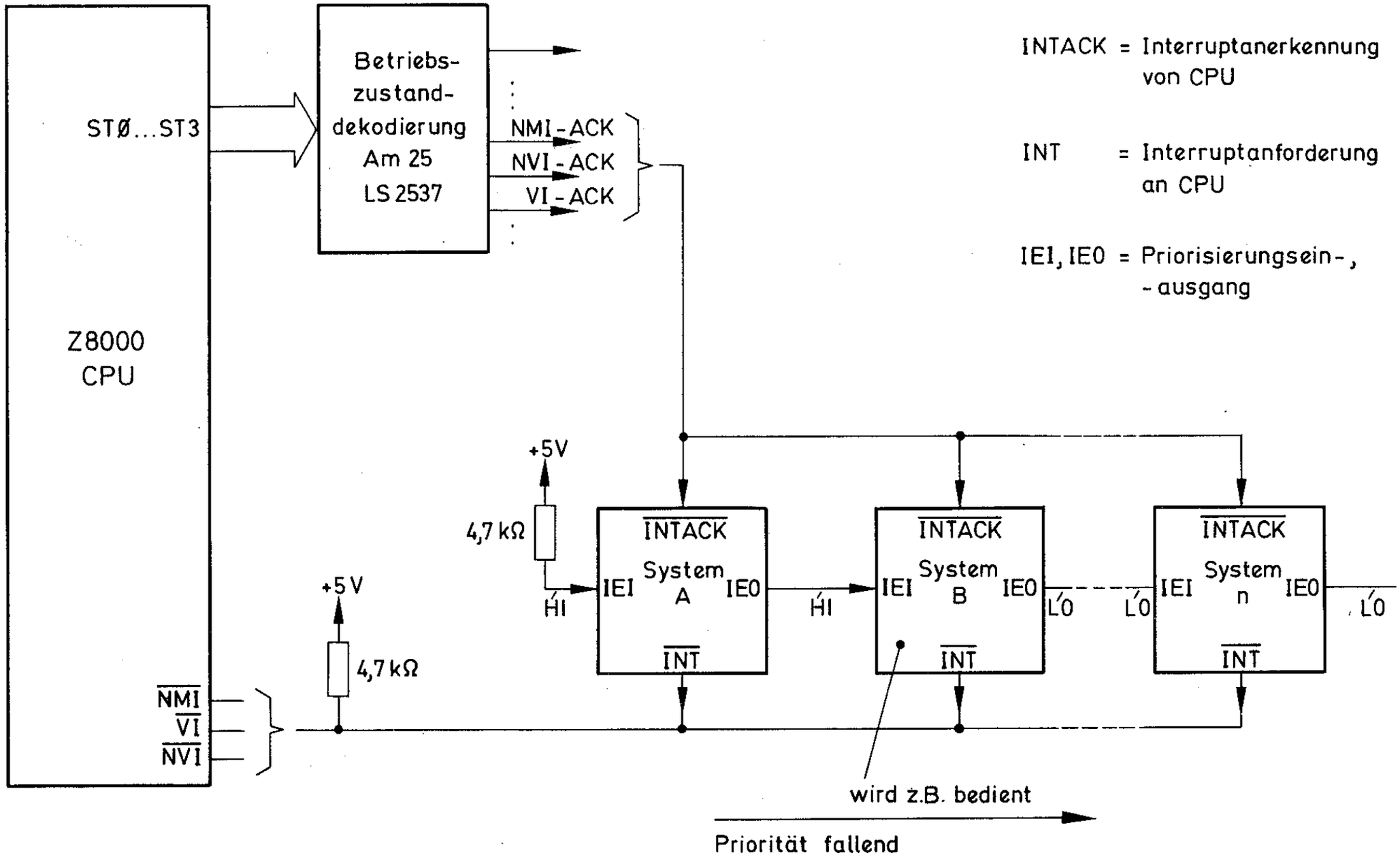
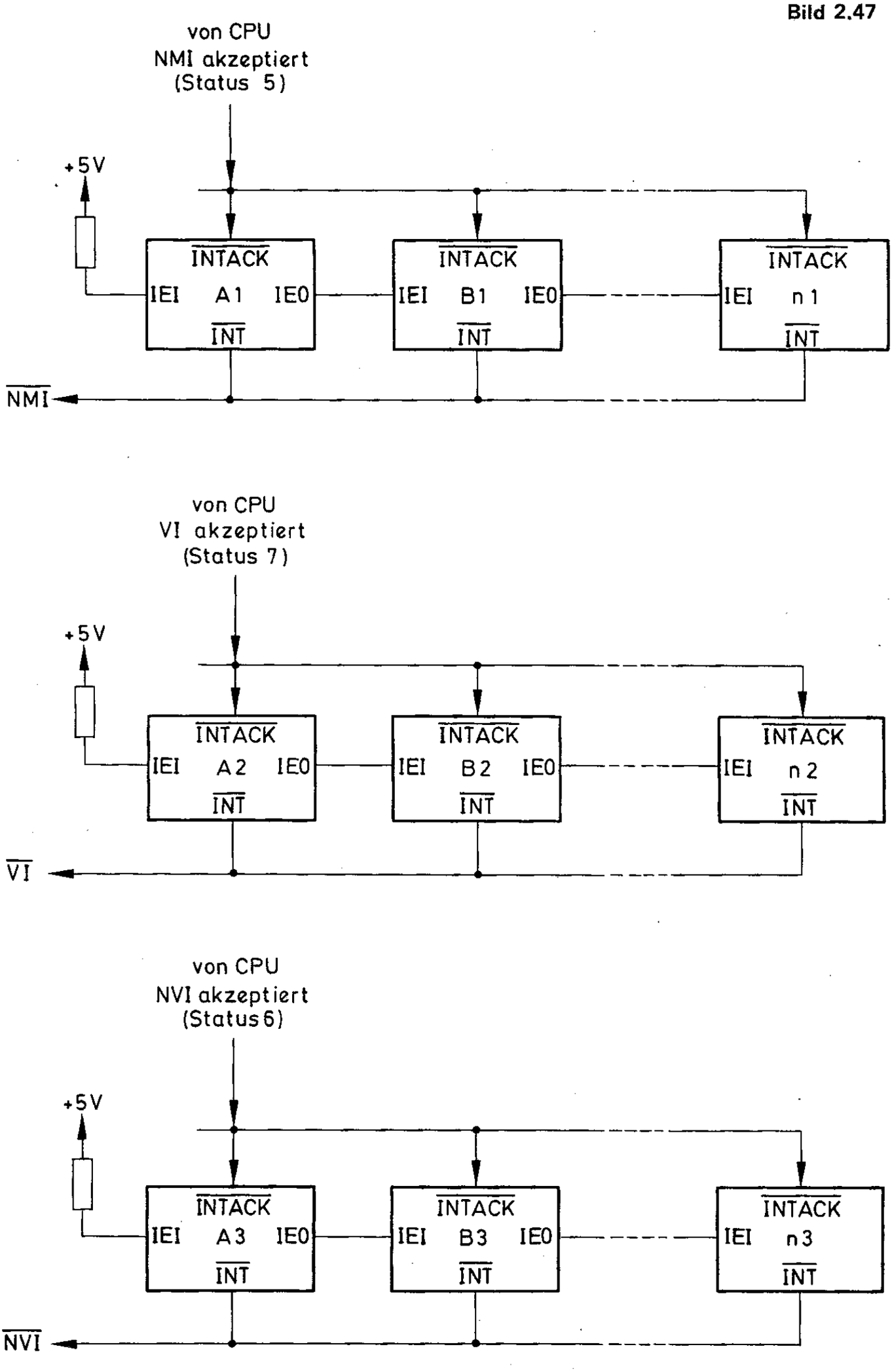

Z8000
CPU
STØ...ST3
Betriebs-
zustand-
dekodierung
Am 25
LS 2537
NMI-ACK
NVI-ACK
VI-ACK
INTACK = Interruptanerkennung
von CPU
INT = Interruptanforderung
an CPU
IEI, IEO = Priorisierungsein-,
-ausgang
+5V
4,7 kΩ
+5V
4,7 kΩ
INTACK
IEI
System
A
IEO
INT
HI
HI
INTACK
IEI
System
B
IEO
INT
LO
LO
INTACK
IEI
System
n
IEO
INT
LO
NMI
VI
NVI
wird z.B. bedient
Priorität fallend

Bild 2.46

Bild 2.47


von CPU
NMI akzeptiert
(Status 5)
+5V
INTACK
IEI A1 IEO
INT
INTACK
IEI B1 IEO
INT
INTACK
IEI n1
INT
NMI
von CPU
VI akzeptiert
(Status 7)
+5V
INTACK
IEI A2 IEO
INT
INTACK
IEI B2 IEO
INT
INTACK
IEI n2
INT
VI
von CPU
NVI akzeptiert
(Status 6)
+5V
INTACK
IEI A3 IEO
INT
INTACK
IEI B3 IEO
INT
INTACK
IEI n3
INT
NVI

führt werden, wenn das interne NMI-Flag (nicht im FCW) durch ein externes Ereignis gesetzt wurde. Die beiden maskierbaren Interrupts $\overline{VI}$ und $\overline{NVI}$ sind im FCW-Register an Bit 12 und 11 zu setzen, wenn jemals an den Statusausgängen eine Anerkennung erfolgen soll.

Prinzipiell läßt sich eine "daisy-chain"-Buslogik relativ leicht durch ein paar Gatter selbst bauen. Im gegebenen Fall muß man entscheiden, ob sich das trotzdem lohnt. In den meisten Fällen wird ein Schnittstellenbaustein aus der Z 8000-Familie zu verwenden sein.

In **Bild 2.47** ist das Spektrum der Z 8000-Interruptebenen dargestellt. Diese Netzwerke der jeweiligen Ebene sind von einer CPU zu betreuen. Für einen $\overline{NMI}$ ist es jedoch fraglich, ob eine Priorisierung in mehreren Stufen sinnvoll ist. Einen $\overline{NMI}$ sollte man in der Tat nur für sehr wichtige Ereignisse verwenden, wie z.B. eine Speicher-Notstromeinschaltung bei unvorhergesehenem "Power-off". Das kommt einem Erdbeben auf der Computerebene gleich. Ein Funktionsausfall von Speichern, der von dem Fehlerkorrektur-Baustein AmZ 8160 festgestellt wurde, ist ebenfalls ein $\overline{NMI}$ wert. Die $\overline{VI}$-Ebene ist für Steuerungs- oder vielmehr regelungstechnische Anwendungen sehr gut geeignet, weil die Interrupt-Serviceroutinen sehr spezifisch den jeweiligen Ereignissen angepaßt werden können. Der zu Beginn des Abschnitts beschriebene "Drehvorgang" ist in der hardwaremäßig gestaffelten Priorität auf dieser Ebene mit einem Z 8000-System automatisierbar. Für die niederwertigste Ebene, den $\overline{NVI}$, ist kein spezieller Anwendungsbereich zu nennen. Dieser Interrupt bewirkt auf der ganzen Prioritätsebene die Auslösung eines einzigen Typs einer Serviceroutine. Es gilt eine geeignete Anwendung zu suchen, die für eine gewisse Zahl von Ereignissen den gleichen Typ einer Serviceroutine benötigen. Wie niedrig die Ebene eines $\overline{NVI}$ ist, zeigt das Flußdiagramm in **Bild 2.48,** das die Anerkennung eines $\overline{NVI}$ zum Ziel hat. Vier Verzweigungen sind zu durchlaufen, bis die Anerkennung erfolgen kann. Bei der Abfrage des NVIE-Flags während des ersten Befehlholzyklus, nach dem der $\overline{NVI}$ aufgetreten ist, wird mit diesem IF 1 unmittelbar weitergearbeitet, wenn dieses Flag im FCW Null war. Mit dem Pfeil auf dem Fluß zwischen der dritten und vierten Verzweigung soll die ständige Unterbrechbarkeit durch einen $\overline{NMI}$ angedeutet werden. Für die extremen Leute unter den Interruptsspezialisten sei noch angemerkt, daß der Segment-Trap-($\overline{SEGT}$) Eingang des Z 8001 als weiterer, genereller Interrupteingang benutzt werden kann, wenn keine MMU im System eingebaut ist. Dies sei jedoch nur am Rande erwähnt.

Hardwaretechnisch ist zum $\overline{NMI}$ und $\overline{NVI}$ noch folgendes zu bemerken. Die Z 80-CPU springt bei einem akzeptierten $\overline{NMI}$ automatisch auf die Programmzählerstelle 0066H. Beim Z 8000 ist das in der PSAP-Sprungtabelle festgelegt. Man kommt um die Kennung als auslösendes Element herum. Wird ein $\overline{NMI}$ akzeptiert, muß für die CPU am Datenbus D0 . . . D 15 nur eine Pseudo-Kennung als eine Art Status im Anerkennungszyklus zu lesen sein. Bei einem $\overline{VI}$ ist das schon etwas komplizierter.
Wie wird beispielsweise ein $\overline{VI}$ von einem Schnittstellenbaustein mitorganisiert? Als Beispiel sei der Z 8030 SCC herangezogen. Nach der Anerkennung eines bestimmten Interrupts für den Baustein kann die CPU im Anerkennungszyklus die Kennung aus einem seiner Steuerregister auslesen. Das Kennwert des SCC bildet sich aus mehreren Teilen. Das Format eines Steuerregisters hat folgendes Aussehen:

AD 7	AD 6	AD 5	AD 4	AD 3	AD 2	AD 1	AD 0
4-bit Kennung				**Status**			ϕ **immer**

Das 4-bit-Kennungsfeld zeigt die Sprungziele zu einer Vektorgruppe in oktalen Gruppen an. Steht im Kennungsfeld 0000, sind die ersten 8 Vektoren (Vektor 0 . . . 7) gemeint. Steht 0001 drin, sind die Vektoren 8 . . . 15 gemeint. Bei 0011 sind die Vektoren 16 . . . 23 gemeint usw. Diese 4 Bits sind unmittelbar Werte für die Kennung, die an die CPU gerichtet sind. In dem 3-bit-Statusfeld wird der Kanal und die Datenregister spezifiziert, die bei der Serviceroutine im SCC betroffen sind (der SCC-Baustein hat zwei unabhängige Voll-Duplex-Kanäle). Diese 3 Bits gehören nicht zur eigentlichen Kennung, denn an Hand der Sprungtabelle ist nur ein einziger Vektor zulässig, der genau auf einen Baustein zugeschnitten ist. Man setzt einen Wert ein, um die Kennung komplett zu machen. Der 3-bit-Code spezifiziert auch die Beschaffenheit eines Kanalspeicherblocks, z.B. ob er leer ist. Man muß sich eben überlegen, was die CPU mit und über den SCC machen soll. Der Wert AD 0 ist immer Null. Der Status wird von der CPU in der spezifischen Interruptroutine gelesen und sodann auch kanalrichtig interpretiert. Es kann auf eine Eingabe über Kanal A eine Ausgabe über Kanal B erfolgen. Während der Interruptroutine kann die CPU jedes Steuer- und Datenregister des SCC lesen. Schreiben und lesen mit dem SCC geschieht während der Interruptroutine logischerweise mit E/A-Befehlen der CPU. Über die Adreßausgabe von AD 1 . . . AD 5 kann die CPU auf jeden einzelnen der 64 Datenregister des SCC zugreifen. Die eigentliche Aufgabe der Interruptroutine besteht nun darin, die aktuellen Zustände eines Prozeßteils nach einem geeigneten Regelgesetz zu verarbeiten und die Ergebnisse z.B. über den Kanal B des SCC nach außen zu den Stellgliedern des Regelsystems zu übergeben.

Man legt also irgend einen Sprungvektor fest, z.B. 30H. Dieses Kennungs-Wort wird nach der Anerkennung eine Adresse in den Programmzähler der CPU initiieren, die genau diesen Baustein über eine Serviceroutine bedient. Auf dem oberen Busbyte AD 8 . . . AD 15 sind entweder Nullen oder Einsen (mit pull-up-Widerständen) festzulegen. Die 16-bit-Kennung hat entsprechend den Wert 0030H oder FF30H. Einer dieser Werte wird im Anerkennungszyklus in den Systemstapel übernommen. Die eigentliche Vektoradresse wird folgendermaßen gebildet:

Kennung (unteres Byte) x 2 + 3CH (bzw. 1EH)

Der Wert 3CH für die neue Programmzähleradresse einer VI-Serviceroutine des Z 8001. Der Wert 1EH entsprechend der für den Z 8002. Die Verdoppelung der Kennung ist für beide CPUs gleich. Somit ergibt sich als Vektoradresse für den Z 8001:

30H x 2 + 3CH = mm9CH entspricht 24. Vektor

und für den Z 8002:

30H x 2 + 1EH = mm74H entspricht 48. Vektor

Diese Adreßberechnung führt die CPU automatisch durch. Sie ist nur für VIs nötig. Bei allen anderen Ereignissen wird direkt in die Werte in der PSAP-Tabelle gesprungen

In **Bild 2.49** ist die interne Erzeugung eines Signalmusters dargestellt, das ein System unter Service einem niedriger priorisierten System in der Kette übergibt, das ebenfalls eine $\overline{\text{INT}}$-Anforderung angemeldet hat. Dieses asynchrone Anforderungs-/Anerkennungs-Schema ist, wie schon erwähnt, für alle drei Arten der Interrupts gültig. In einer **Tabelle** sind alle Signale dieses Interruptprotokolls einzeln aufgeführt. Ist eine Anerkennung durch $\overline{\text{INTACK}}$ erfolgt, wird in dem Priorisierungsschema immer nur ein System bedient. Wird das System Nr. 15 bedient, ist es jedoch nicht möglich, daß die Bedienung von dem höherpriorisierten System Nr. 3 unterbrochen wird. Das wäre auch nicht sinnvoll. Im FCW-Status läßt sich das für eine VI- und NVI-Serviceroutine unterbinden,

Tabelle der Signale für das Statusdiagramm einer Interrupt-Bedienung in der Kette

IEI Dieses Eingangssignal eines Interrupt fähigen Bausteins zeigt diesem in der Kette gelegenen an, ob eine Bedienung einer Interrupt-Anforderung möglich ist. IEI wird als ein Verknüpfungspunkt in der gesamten Kette angesehen, die für eine bestimmte Interruptebene gilt. Ein Hi an IEI zeigt dem Baustein an, daß grundsätzlich eine Interrupt-Anforderung die Aussicht auf eine Bedienung der CPU verspricht, weil keine höher priorisierte Anforderung vorliegt. Ein Lo an IEI zeigt hingegen an, daß ein Baustein an einer höher priorisierten Stelle der Kette eine Anforderung über $\overline{INT}$ ausgesendet hat.

IEO Dieses Ausgangssignal an einem Interrupt fähigen Baustein wird von diesem dann aktiviert, wenn die Bedingungen für eine Aktivierung erfüllt sind (IEI = Hi, IE = Hi, IP = Hi, IUS = Lo).
IEO wird Lo gesetzt und zugleich der $\overline{INT}$-Ausgang auf Lo aktiviert. IEI, IEO ist die Ein-/Ausgangskettenverbindung in Bezug auf die Priorisierung in der Kette. IEO eines Interrupt fähigen Bausteins zeigt auf IEI des nächst niedrigeren in der Kette. Im Ruhezustand, wenn keinerlei Anforderungen vorliegen, haben IEI und IEO immer das gleiche Potential Hi.

$\overline{INT}$ Dieser Ausgang eines Interrupt fähigen Bausteins dient zur Übermittlung der Anforderung auf eine Interruptbedienung von der CPU.

Jeder Interrupt fähige Baustein arbeitet intern mit einer Reihe von Signalen bzw. Statuswerten, von denen IE, IP, IUS und DLC als die wichtigsten gelten.

$\overline{INTACK}$ Dieser Eingang zeigt dem Interrupt fähigen Baustein an, daß die CPU eine Interruptanforderung auf einer Ebene akzeptiert hat.

IE (Interrupt enable). Dieses Bit betrifft die Freigabe des internen Flagbits, das extern durch ein Strobe-Signal gesetzt wird und damit dem Baustein ankündigt, daß über ihn ein Transfer (in einer Serviceroutine inbegriffen) anzufordern ist. Ist IE gleich Lo, kann das Flagbit über Strobe nicht gesetzt werden und somit kein Interrupt erfolgen. Ist IE gleich Hi, kann ein Interrupt angefordert werden. Dieses Bit kann von der CPU in einem Z-Peripheriebaustein gesetzt werden.

IP (Interrupt pending). Dieses Bit zeigt intern der Interruptlogik des Bausteins eine Anforderung an (Strobe-Signal extern aktiv). Das Interruptsignal $\overline{INT}$ wird gesetzt, wenn die notwendigen Bedingungen dafür erfüllt sind (IE, IEI, IUS, $\overline{INTACK}$). Dieses Signal kann von der CPU gelesen werden und dient somit auch als Entscheidungsbit bei einer Programmschleife.

IUS (Interrupt under service). Dieses Bit zeigt der Interruptlogik des Bausteins an, daß ein Interrupt schon in Bearbeitung oder in Anforderung ist. IUS bezieht sich nur auf den Baustein intern und ist mit den Strobe-Flags verknüpft. Da ein Interrupt fähiger Baustein mehrere Quellen einer Interruptanforderung besitzen kann, muß dieses Bit anzeigen, daß für jede Einzelquelle eine Anforderung angestrebt werden kann. Jedes einzelne Strobe (externe Auslösung) hat zur internen Bearbeitung ein IP-, IE- und IUS-Bit zur Verfügung. Alle IUS-Bits müssen Lo sein, damit eine Anforderung Aussicht auf Erfolg hat. Ist ein IUS-Bit auf Hi gesetzt, zeigt das eine Bearbeitung an. Die anfordernde Quelle wird somit verzögert, solange nicht jedes IUS-Bit Lo-geworden ist.

DLC (Disable lower chain). Ist dieses Bit auf Hi gesetzt, wird der Ausgang IEO auf Lo gesetzt, um eine Anforderung jedes niedriger priorisierten Bausteins zu unterbinden.

Ein Interrupt fähiger Baustein kann einen oder mehrere Vektoren zur Kennung bei einem Anerkennungszyklus ausgeben, die er vorher in seinen Vektorregistern eingeschrieben bekam. Jede Interrupt-Quelle bekommt einen Vektor zugewiesen. Jede Interrupt-Quelle mit Vektor hat ein Vektorstatusbit VIS zur Kontrolle (VIS = Lo → Vektorzuweisung für Interrupt-Quelle; VIS = Hi → Vektor kann verändert werden). Mit dem Statusbit NV (No vector) wird definiert, ob der Interrupt fähige Baustein eine Kennung für einenNVI im Anerkennungszyklus aussendet (NV = Hi) oder bei einer NVI-Anerkennung die Kennung im Befehl selbst steht oder von einer anderen Logik stammt (NV = Lo).

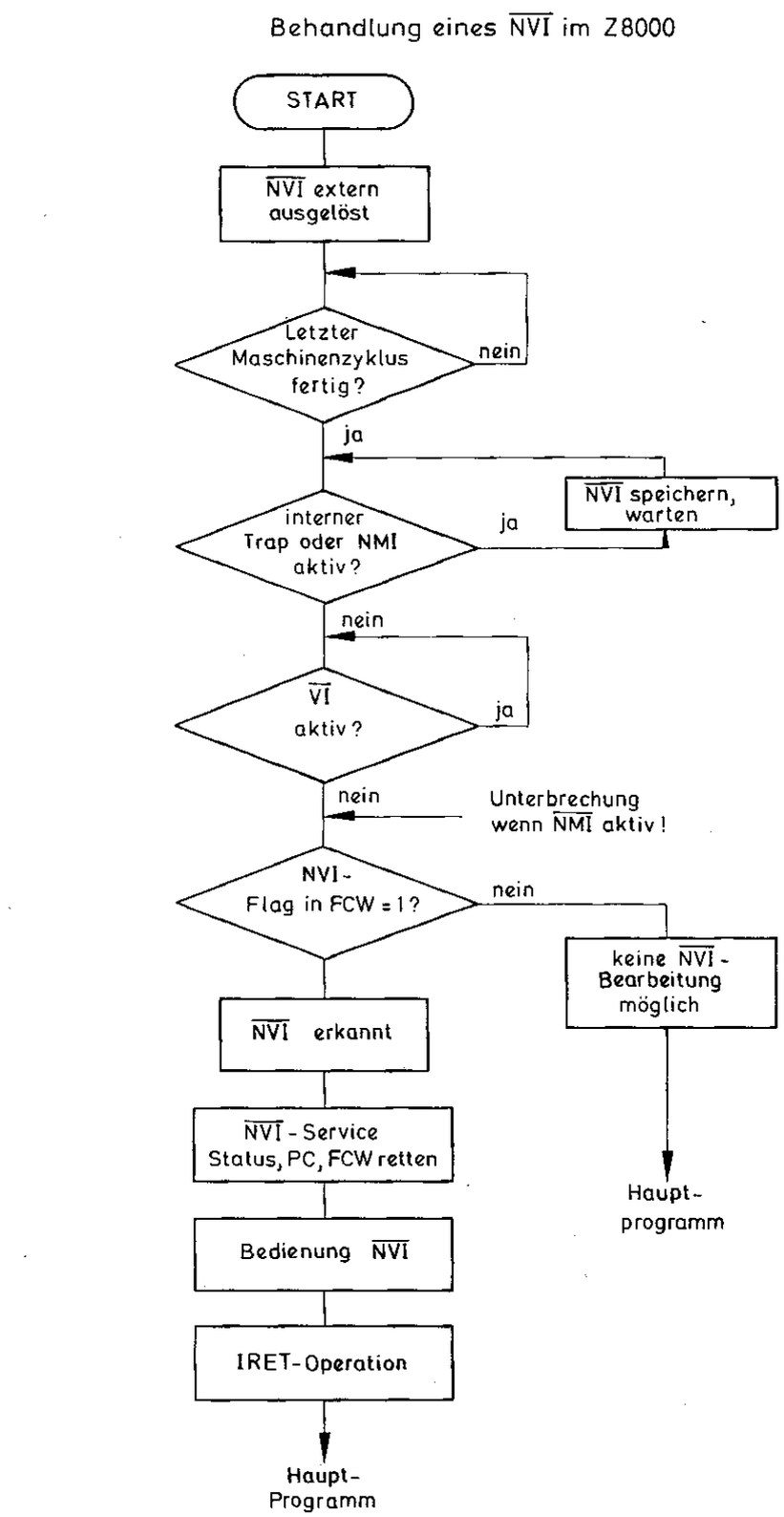


Bild 2.48

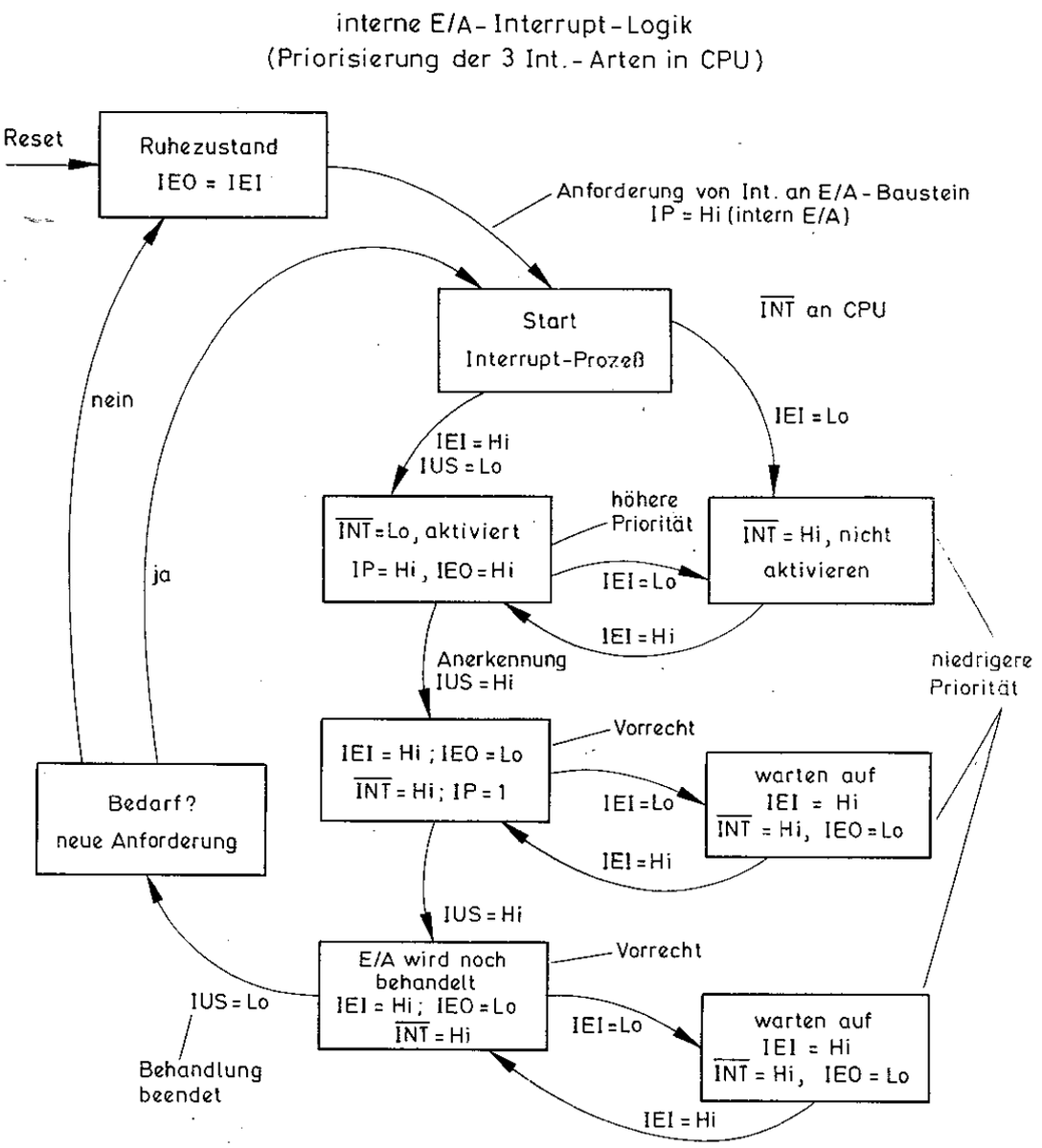


Bild 2.49

indem man die Flags für diese Ebenen nicht setzt. Der FCW-Status in der Sprungtabelle SSXX38H gilt für alle VIs. Nur ein Wort $\overline{\text{NMI}}$ oder Trap kann eine Unterbrechung eines VI-Service herbeiführen. Wenn ein $\overline{\text{NMI}}$ in einem großen System erscheint, ist ohnehin der Teufel los. Eine Serviceroutine wird nach dem genannten FCW-Status grundsätzlich stetig ausgeführt und mit IRET beendet, bevor die nächste Anforderung bedient wird. In der Zwischenzeit muß ein anderes System auf die Beendigung oder die Hardware-Freigabe IEI gleich Hi warten. Erst nach einem Rücksprung kann eine neue Bedienung erfolgen. Wenn mehr als 10 Systeme auf einer Ebene verkettet sind, ist durch die IEI-IEO-Laufzeit eine Verzögerung von IEI eingetreten, die einen oder mehrere zusätzliche WAIT-Zyklen der CPU notwendig machen, damit die Kennung von der CPU ordnungsgemäß übernommen werden kann.

2.10 BUSANFORDERUNG VON EXTERNER OPERATIONS-EINHEIT (DMA)

Wie die bisherige Darstellung der technischen Einzelheiten der Z 8000-CPU zeigte, ist sie zu den sehr schnellen Prozessoren aller erfaßbaren Typen zu zählen. Das ist hard- und softwaremäßig in gleicher Weise bedingt. Die routinemäßige Bewegung von Daten läßt sich entsprechend schnell durchführen. Der Z 8000-Befehlssatz beinhaltet schnelle Speicher zu Speicher Block-Transfer-Befehle, Test-Byte-Strings, Such- und Vergleichsbefehle, ferner Block-Transfer-Befehle zwischen Ports und Speichern. Man könnte fragen, ob ein DMA-System im einzelnen noch mehr zu bringen vermag. Diese Frage wäre berechtigt, gäbe es den Super DMA-Baustein Z 8016 von AMD nicht. Dieser Transferbaustein für direkten Speicherzugriff kann mit einer Geschwindigkeit von 2 MByte/s Daten zwischen Speicher und Port übertragen. Er kann im gleichen logischen Adreßfeld von 8 MByte arbeiten, wie die Z 8001-CPU. Soll der Z 8016 mit der CPU zusammenarbeiten können, muß die CPU zuweilen ihre Priorität auf den Bus abgeben können. Dies kann sie, wie viele andere Prozessoren auch. Wird eine hardwaremäßige Anforderung auf den Bus an den $\overline{\text{BUSRQ}}$-Eingang der CPU gegeben, zeigt sie am Anerkennungsausgang $\overline{\text{BUSAK}}$ zu Beginn des nächsten Maschinenzyklusses an, daß sie 70 % ihrer Bausteinanschlüsse abgeschaltet hat. Es wäre in der Tat falsch, wenn man annimmt, daß die CPU nur wegen DMA-Transfers diese Einrichtung besitzt. Der DMA-Betrieb mit einem passenden Steuerbaustein ist zwar wichtig; es gibt jedoch noch andere Gründe für eine externe Busanforderung. Wie im ersten Kapitel angesprochen ist, hat der Z 8000 eine klare technische Option auf den Einsatz in einem Multiprozessorsystem. Das bedeutet grundsätzlich den freien Zugriff aller verketteten CPUs auf gemeinsame Resourcen wie Speicher, externe Rechenwerke und Verbindungsbusse. Soll ein verteilter Zugriff im System möglich sein, muß das auch steuerungsmäßig richtig in den Griff zu bekommen sein. Das funktioniert nur über ein praktisches Abschalten der Busleitungen zur CPU. In **Bild 2.50** ist dargestellt, wie eine Busanforderung in der CPU signaltechnisch behandelt wird. Im ersten Taktzyklus eines beliebigen Maschinenzyklusses wird die Synchronisation eines aktivierten $\overline{\text{BUSRQ}}$ eingeleitet. Ist die Operation ausgeführt, wird der $\overline{\text{BUSAK}}$-Ausgang nach dem letzten Maschinentakt der Befehlshol- oder ausführungs-Operation auf Lo gezogen. Praktisch mit keiner Verzögerung ist der gesamte CPU-Bus (AD-Leitungen, Zeitsteuersignale, Betriebszustandsanzeige und Segmentadreßausgänge nach Bild 2.7) inaktiv und hochohmig. In diesem Zustand hat jeder betroffene Ausgang eine Impedanz von etwa 50 kΩ. Der Anforderungseingang $\overline{\text{BUSRQ}}$ muß so lange auf Lo gehalten werden, bis die Transaktion eines anderen Elements (CPU oder DMA) beendet ist. Wenn die Anforderung auf Hi springt, übernimmt die CPU wieder die normale Aktivität und arbeitet an der Stelle weiter, an der sie stehengeblieben ist. Es treten keinerlei Informationsverluste in der CPU auf. Der $\overline{\text{BUSAK}}$-Ausgang wird nach einem Taktzyklus wieder Hi. Zwei Taktzyklen später beginnt der normale Operations-Modus.

Die hardwaremäßige Busverbindung von DMA-fähiger Peripherie zeigt **Bild 2.51.** Das Prädikat DMA-fähig bezieht sich auf Entlastungs- oder Peripherieprozessoren, die eine genügend hohe eigene Intelligenz besitzen müssen, um CPU-Transaktionen übernehmen zu können. Sie müssen mindestens über das R/$\overline{\text{W}}$-Signal und eine Unterscheidung zwischen Speicher und Port verfügen. Für die Z 8000-Systembausteine schließt das auch die Signale $\overline{\text{AS}}$ und $\overline{\text{DS}}$ und daraus ableitend einen gemeinsamen Adreß-/Datenbus mit ein.

Die Technik der Busverknüpfung wird

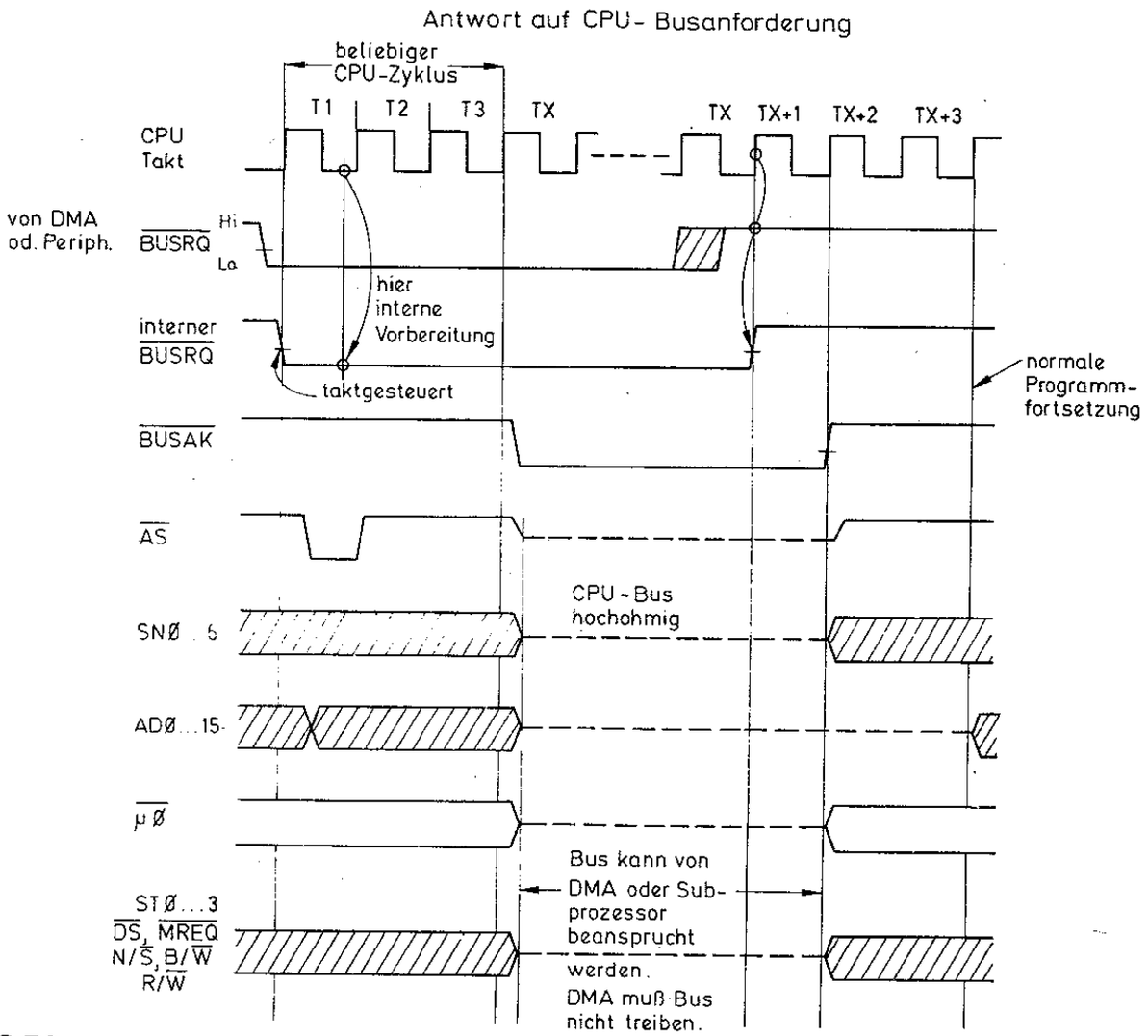


Bild 2.50

wieder nach dem "daisy-chain"-Prinzip aufgebaut. Das Signal $\overline{BAO}$ (Busanerkennungs-Ausgang) des Systems A_1 bleibt offen. Bis zur CPU hin sind die Anforderungsleitungen $\overline{BRQ}$ der einzelnen Systeme bidirektional, damit jedes System die Aktivität eines anderen mitverfolgen kann. Ein System meldet keine Anforderung an, wenn $\overline{BAO}$ gleich Hi ist, oder besser, wenn $\overline{BAI} = \overline{BAO}$ ist ($\overline{BAI}$ = Busanerkennungs-Eingang). Auch in der Systemkette für eine Busanforderung ist eine Priorität mitverknüpft. Das System A_1 kann eine Anforderung anmelden, der zu allererst stattgegeben wird. Die Hierarchie ist eben ein typisches Kennzeichen eines "daisy-chain"-Busses.

Worin liegt nun eine Anforderung eines externen Systems auf den CPU-Bus begründet? Grundsätzlich sind zweierlei Arten zu unterscheiden: Entweder übergibt die CPU dem DMA-Baustein einen Befehl, daß sie einen Transfer übernehmen soll, worauf der DMA mit einer Anforderung antwortet, oder eine Anforderung auf den Bus entsteht, gleichsam wie ein Interrupt, völlig unvorhergesehen, weil der Nachbarprozessor in einem Multi-System auf den Speicher zugreifen will. Der Unterschied ist deshalb grundlegender Art, weil beim DMA z.B. ein Transfer in einer Interrupt-Serviceroutine fest einprogrammiert ist, den die CPU übergibt. Greift ein Nachbarprozessor auf ein gemeinsames Resource zu, kann dieser die gleich hohe Verfügungspriorität wie die CPU besitzen, die momentan vom Bus abgetrennt wird. Dies liegt in der Philosophie des Systems begründet.

Betrachtet man sich die beiden "daisy-chain"-Konfigurationen in Bild 2.46 und 2.51, ist wirklich nur ein kleiner Unterschied anzumerken. Das Anerkennungssignal $\overline{BUSAK}$ der CPU wird

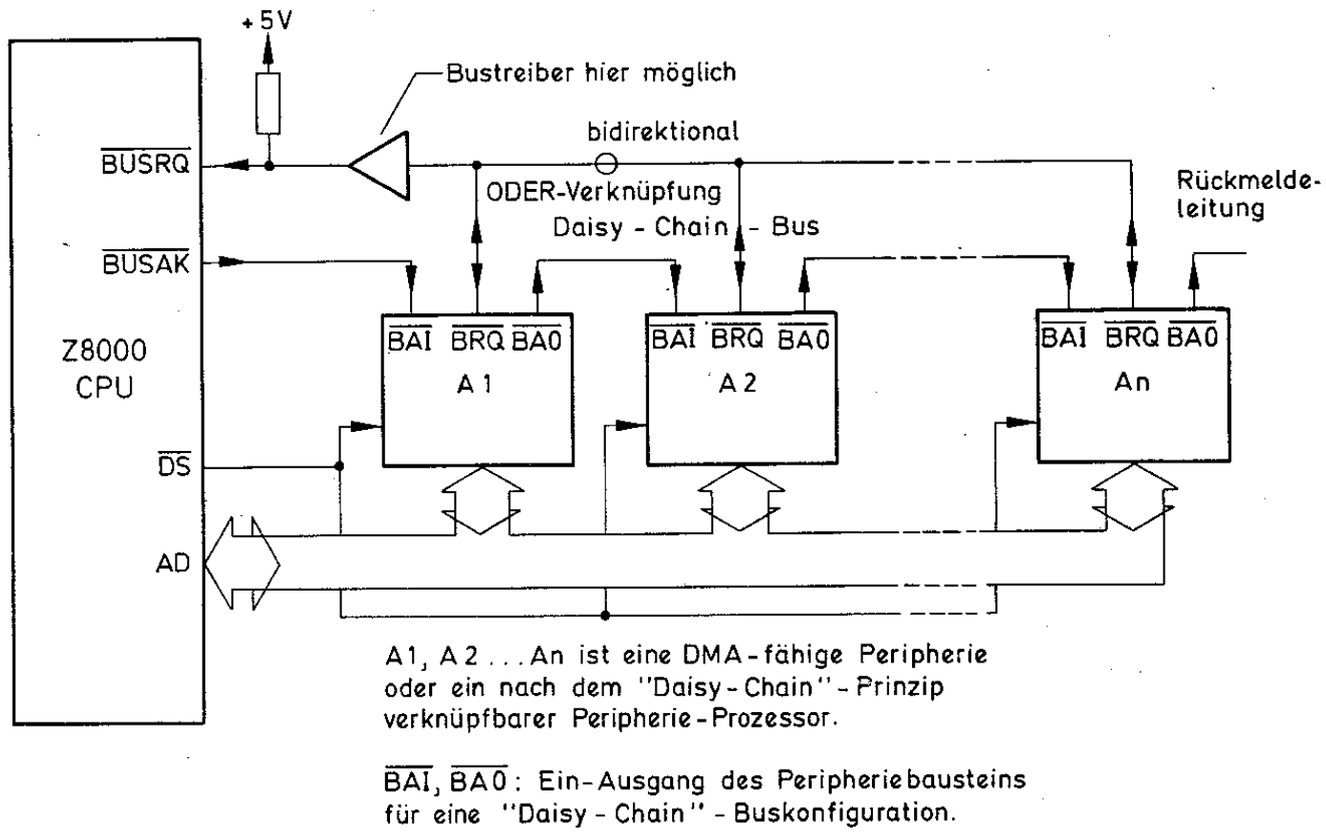


Bild 2.51

direkt an den Anerkennungseingang BAI gegeben, wogegen INTACK auf alle Bausteine parallel gegeben wird. Die Priorisierung des höchsten Systems im INT-"daisy-chain" wird am IEI statisch durch +5 V festgelegt. Würde man es über die Zustandsdekodierung festlegen, könnte beim Umschalten des Status eine unbeabsichtigte Auslösung auf den IEI-Eingang gegeben werden, was sicherlich auf die Dauer sehr ärgerlich wäre. BUSAK kommt von der CPU auf keinen Fall unbeabsichtigt. Das ist der Grund. Die Anforderungssignale INT sind praktisch mit der Anforderung Busrequest BRQ identisch. Die Anerkennungs-Ein- und -Ausgänge BAI und BAO sind mit IEI und IEO ebenfalls identisch.

In **Bild 2.52** ist das Protokoll bzw. Flußdiagramm einer Busanforderung dargestellt. Wird System A2 in Bild 2.51 eine Anforderung benötigen, wird zuerst der bidirektionale BRQ-Eingang abgefragt. Wenn das Signal Lo ist, kann A2 den Bus noch nicht beanspruchen. A2 muß warten, bis BAI inaktiv (Hi) ist. Erst dann kann ein "Angriff" von A2 auf den Bus gestartet werden.

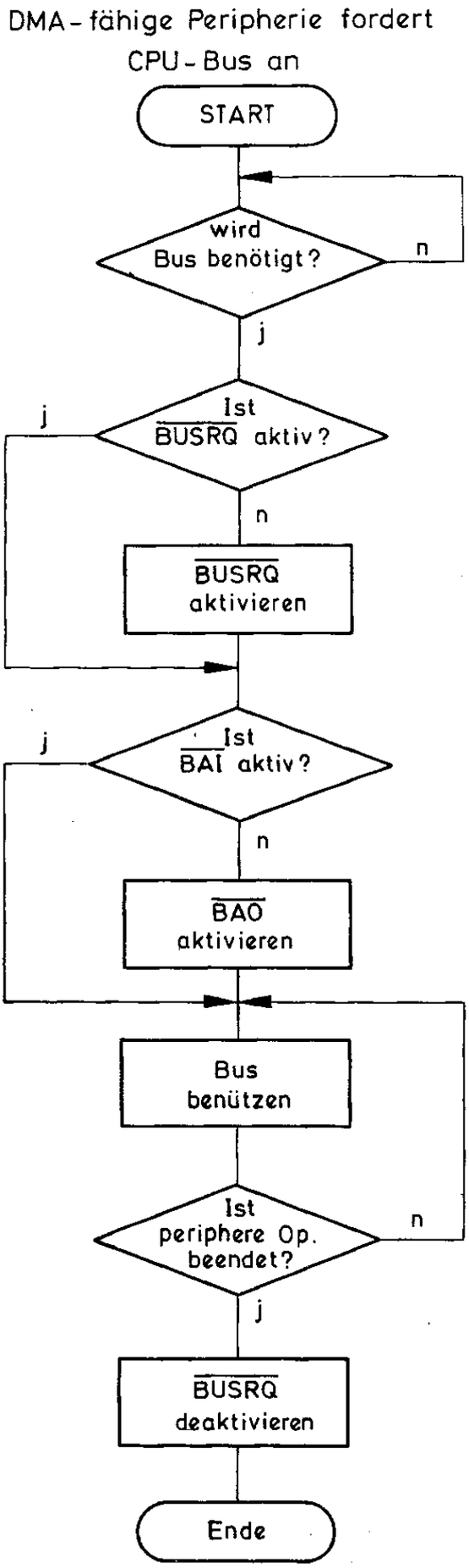

DMA-fähige Peripherie fordert
CPU-Bus an
START
wird
Bus benötigt?
n
j
Ist
BUSRQ aktiv?
j
n
BUSRQ
aktivieren
Ist
BAI aktiv?
j
n
BAO
aktivieren
Bus
benützen
Ist
periphere Op.
beendet?
n
j
BUSRQ
deaktivieren
Ende

Bild 2.52

2.11 MULTIPROZESSORFÄHIGKEIT (HARD- UND SOFTWARE-MÄSSIG)

Multiprocessing, multitasking, parallelprocessing, multi-micro-processor-facilities sind die Reizworte über Computerarchitekturen der Zukunft, die aber in den 80er Jahren sicher schon sehr diskrete Formen annehmen werden. Man liest in den Fachmedien amerikanischer und auch deutscher Herkunft immer öfter davon. Vereinzelt sind reale Umsetzungen einer entsprechenden Architektur bereits erfolgt. Bis zu diesem Abschnitt wurden Multiprozessor-Systeme wenigstens angedeutet.

Multiprocessing ist die Form der koordinierten Zusammenarbeit mehrerer CPUs auf einer priorisierten oder neutralen Ebene. Man unterscheidet generell zwischen Multiprozessor-Systemen, die grundsätzlich für verschiedene Tasks jeweils eine Prozessoreinheit verwenden oder solche, deren Ziel eine echte parallele Verarbeitung ist. Bei den ersteren besteht die Zusammenarbeit dieses Systems verteilter Intelligenz darin, den Host-Prozessor von vielerlei Peripherie- und Routine-Tasks zu entledigen. Diese Teilbereiche zum Betrieb eines großen Systems wie z.B. Schnittstellenverwaltung, Disk-Drive-Organisation oder Analogdaten-Übernahme (Acquisitions-System) werden entweder von autonomen Subsystemen oder Single- bzw. Subprozessoren ausgeführt, die über einen Zugriff auf allgemeine Resourcen verfügen. Das Betriebssystem des Multirechners ist in ausschließliche Zuständigkeiten aufgesplittet, deren genannte Teilbereiche nicht mehr im eigentlichen "main-frame-Betriebssystem enthalten sind. Der Hostcomputer ruft mit einem Befehl nur einen bestimmten Teilbereich auf, ohne ihn zu bearbeiten. Über die bereits beschriebene Bus-Anforderung werden der Host-CPU Daten zur direkten Bearbeitung im Hauptsystemspeicher zur Verfügung gestellt. Die Host-CPU braucht sich hierum nicht mehr zu kümmern. In einem reinen Monoprozessorsystem erledigt die CPU solche Routinen in parallelen Tasks. Das eigentliche Anwenderprogramm obligt immer noch der ausschließlichen Bearbeitung der Host-CPU.

Ein Parallelrechner hingegen bearbeitet ein großes Hauptprogramm unter der Zuhilfenahme von Unterprogrammen, die von speziellen Prozessoren quasi parallel bearbeitet werden. Die etwas unklare Ausdrucksweise deutet hier an, daß die Variationsfähigkeit einer solchen Bearbeitung relativ groß ist. Die Bearbeitung ist nicht nur quasi, sondern echt parallel, wenn die Initialisierung der einzelnen Unterprogramme bereits erfolgt ist und nur noch das Startzeichen für den Rechenbeginn gegeben werden muß. Es kommt hier auch sehr auf die Systemphilosophie an, die der Parallelrechner repräsentiert. Besitzt er, wie im vorhergehenden Fall des Multiprozessor-Systems, mit seinen Subsystemen eine Slave-Struktur bei der Parallelverarbeitung, ist eine parallele Bearbeitung nicht immer gegeben. Es hängt ja auch vielfach vom Anwenderprogramm selbst ab. Sind die Unterprogrammesdes aufgesplitteten Hauptprogramms von einer zeitlichen Sequenz unabhängig, ist eine Parallelverarbeitung sicher gut möglich. Ferner kommt es darauf an, ob die Subprozessoren der Kern eines autonomen Subsystems sind und nach einer vorhergehenden Grundinitialisierung einzelner Unterprogramme nur noch einige Daten benötigen, die sie sich per DMA aus dem main-frame-RAM holen. Um so besser ausgebaut ein Subsystem ist, um so weniger wird es die Multibusverbindungen beanspruchen müssen. Es hängt auch von der Initialisierung ab, die ihm zuteil wird. Grundsätzlich kann man jedoch sagen, daß ein Parallelrechner nach einer Master-Slave-Struktur mit autonomen Subsystemen eine echte und teilweise sehr effektive parallele Verarbeitung ermöglicht, so daß ein großes Programm gleichzeitig von mehreren CPUs bearbeitet wird oder mehrere CPUs gleichzeitig an mehreren Anwenderprogrammen ar-

beiten. Überwiegend bestehen solche parallele Tasks aus schwierigen, zeitintensiven Mathematik-Programmen. Der Host-Prozessor stellt die Ergebnisse danach oder noch während der Parallelrechenphase zusammen. Die Subprozessoren zeichnen sich durch besondere Fähigkeiten in einzelnen Disziplinen aus.

Das Problem bei Multiprozessor-Systemen oder Parallelrechnern liegt nun leicht erkennbar darin, wie sich die Subprozessoren untereinander synchronisieren, daß entweder ein geregelter Zugriff auf den Hauptspeicher erfolgt oder generell eine leicht koordinierbare Initialisierung durch einen Host-Prozessor erfolgen kann. Einen sehr zukunftsweisenden Betrieb zeigt die Verbindung der Z 8000-CPU mit der Z 8000-EPU auf, worauf bereits in Abschnitt 2.7 eingegangen wurde. Dies ist jedoch nur ein Teilaspekt der Multiprozessorfähigkeit des Z 8000-Systems, wobei hier der Schwerpunkt bei der EPU liegt.

Was in diesem Abschnitt gemeint sein soll, ist die eigentliche und ausschließliche Multiprozessorfähigkeit der CPU selbst. Hier soll dem noch immer zweifelnden Leser zum mehrfachen mal dargelegt werden, was diese CPU wirklich für ein Kaliber ist.

Der Z 8000 ist der einzige 16-bit-Prozessor, der über echte, effektive Hilfsmittel zum Aufbau eines Multiprozessor-Systems oder Parallelrechners in vielfältigen Variationen verfügt.

In der Mikroprozessor-Szene gibt es in der Tat keinen vergleichbaren Prozessor mit ähnlichen Eigenschaften. Was diese Hilfsmittel hardware- und softwaretechnischer Art betrifft: Sie dienen dazu, die teilweise schwierige Synchronisation eines Transfers zwischen zwei beteiligten Prozessoren eines Multi-System-Netzwerks in eindeutig erfaßbare, gut machbare Bereiche zu rücken. Wann ist der eine oder andere Typ dran, aus dem Hauptspeicher oder von der Host-CPU Daten zu übernehmen bzw. auszugeben? Diese in ihrer Komplexität nicht zu unterschätzenden Details treten bei der Entwicklung solcher extensiver Rechnerstrukturen auf.

Die Variationsfähigkeit und Problematik der Architektur von Multiprozessor-Systemen wurde in einem umfassenden Bericht in der Fachzeitschrift ELEKTRONIK Nr. 17. 1979 unter dem Titel "Systeme mit mehreren Mikroprozessoren" von H. Bardach und D. Hammer beschrieben. Hier kommt zum Ausdruck, wie komplex solche Systeme sein können. Ohne das in diesem Bericht auf ein spezielles System eingegangen wird, sind verschiedene Strukturen genannt, wobei zu symmetrischen und unsymmetrischen Systemen auf die Schwierigkeiten hingewiesen wird. Es heißt hier wörtlich: "Eine variable Aufgabenverteilung bedingt ein sehr komplexes Betriebssystem, um die einzelnen Aufgaben zeitlich und räumlich so zu verteilen, daß keine Konflikte auftreten und der Durchsatz des Systems möglichst groß ist". Es ist hard- und softwaretechnisch ein außerordentlich schwieriges Unterfangen, für variable Aufgabenverteilungen der Prozessoren eine geregelte Zusammenarbeit zu sichern. In Kapitel 7swird hierauf noch einmal eingegangen.

Die Z 8000-Hilfsmittel zum Aufbau eines Multiprozessor-Systems gliedern sich in Hardware und Software.

Für die Hardware gilt:
Separates "daisy-chain"-Interface mit zwei diskreten Anschlüssen am Baustein (für beide Versionen gleich). Der Eingang $\overline{\mu I}$ (Multi-Micro-In) dient zur Übernahme einer Anforderung auf eine Transaktion eines Nachbarprozessors mit einer Resource. Der Ausgang $\overline{\mu O}$ (Multi-Micro-Out) dient zur Ausgabe eines Signals zur Anforderung an einen Nachbarprozessor bzw. einer Resource auf den Bedarf eine Transaktion.

Für die Software gilt:
Vier Befehle sind für ein Handling mit dem Micro-"daisy-chain"-Interface im Befehlssatz inbegriffen.

MBIT (Test Multi-Micro-Bit)
MREQ (Multi-Micro-Request)

MRES (Multi-Micro-Reset)
MSET (Multi-Micro-Set)

Mit dem Befehl **MBIT** kann der $\overline{\mu I}$-Eingang abgefragt werden, ob eine Anforderung eines in der Kette integrierten Subsystems vorliegt. Ist dies der Fall, darf die CPU die Multibusverbindungen der Prozessoren nicht beanspruchen. Die Abfrage nach dem Zustand von $\overline{\mu I}$ geschieht über das Vorzeichen (S-Flag) im FCW. Das S-Flag ist gesetzt, wenn $\overline{\mu I}$ inaktiv (Hi) ist.

Der Befehl **MREQ** ist sehr wichtig, weil er zwei Funktionen erfüllt. Man kann mit seiner Hilfe eine Anforderung auf ein Resource am $\overline{\mu O}$-Ausgang anmelden. Dies geschieht jedoch erst dann, wenn $\overline{\mu I}$ (S-Flag) abgefragt wurde. Wurde $\overline{\mu I}$ abgefragt und als inaktiv (Hi) befunden, wird $\overline{\mu O}$ Lo gesetzt und eine Verzögerungsschleife gestartet. Nach Beendigun der Schleife (Verzögerung) wird das Z-Flag gesetzt. War nach der Verzögerung $\overline{\mu I}$ aktiv (S-Flag gesetzt), kann die CPU auf die Quelle zugreifen. Eine Anforderung (Lo an $\overline{\mu O}$) wird ausgegeben, wenn $\overline{\mu I}$ gleich Hi war (inaktiv). Dieser Befehl ist deshalb wichtig, weil man mit ihm eine Verzweigung in einem Flußdiagramm bilden kann. Mit diesem Befehl läßt sich eine Priorität in der Kette verknüpfter Prozessoren festlegen.

Der Befehl **MRES** ist sehr einfach. Er bringt $\overline{\mu O}$ in den inaktiven Zustand.

Ebenso eindeutig ist der Befehl **MSET.** Er bringt $\overline{\mu O}$ grundsätzlich in den aktiven Zustand (Lo). Dieser Befehl setzt gewissermaßen Zeichen. Wird dieser Befehl angewendet, läßt sich damit ein "Master" im Systemnetz festlegen.

Das "daisy-chain"-Interface zur Synchronisation des Transfers auf Multibusverbindungen muß man erst aufbauen. Wie beim Interrupt-"daisy-chain"-Bus besteht dieser aus vier Leitungen, dessen jeweilig identisches Interface an einer CPU aus zwei Ausgängen und zwei Eingängen besteht. In **Bild 2.53** ist die elementare Form dieses Bustyps dargestellt. Dieser Bustyp bringt wiederum die spezielle Eigenschaft des "daisy-chain"-Busses mit sich, nämlich die stetig fallende Priorität von links. Das bedeutet, daß die CPU (1) zuerst die Multibusverbindung benutzen darf, nach Beendigung die CPU (2) usw. Wenn das auch einen etwas bürokratischen Eindruck macht, man muß hier eben in einer geordneten Reihenfolge für eine Anerkennung einer Anforderung vorgehen, sonst entsteht ein kaum zu entwirrendes Durcheinander. Die CPU (n) kann die Multibusverbindungen beanspruchen, wenn vor ihr keine Anforderung vorliegt und lediglich die Laufzeit des Quittungssignals $\overline{\mu AO}$ an $\overline{\mu AI}$ der CPU (n) anzeigt, daß sie jetzt den Multibus benützen darf. Die höchste Priorität der Kette bildet das System (1), weil $\overline{\mu AI}$ als erster Eingang mit einer allgemeinen Anforderung angesteuert wird. Der Widerstand auf +5 V sollte eingebaut sein, weil im Zustand keiner Anforderung $\overline{\mu AI}$ immer Hi sein soll.

"daisy-chain", dieser Begriff hat es in sich. Sehr geehrter Leser! Bitte verzeihen Sie mir, wenn ich an dieser Stelle ein absolut unwissenschaftliches Beispiel zu diesem Begriff vorstelle. Es scheint mir jedoch sehr plastisch zu sein, sehr zutreffend und obendrein noch ein wenig amüsant.

Stellen Sie sich einen Kasernenhof vor. Eine kleine Truppe ist in Reih' und Glied vor ihrem Anführer angetreten. Nach einigen vorhergehenden Kommandos brüllt der Anführer: "Kompaniiii abzähln". Kaum ist das "n" von abzählen verklungen, wird der erste in der Truppenreihe "eins" brüllen und zugleich (vielleicht) seinen Kopf nach links zu seinem Nachbarn hindrehen. Der Nachbar fühlt sich dadurch gleichsam angeregt, sein "zwei" loszuwerden. Diese Kettenreaktion pflanzt sich in bekannter Weise bis zum letzten Mann in der Reihe fort, der dann (nach Art des "Hauses") noch ein Schlußkommando von sich gibt: "Kompanie 'sowieso' mit ?? Mann angetreten".
Betrachtet man diese Reihe Soldaten mit ihrem Anführer, ist dies mit der Dar-

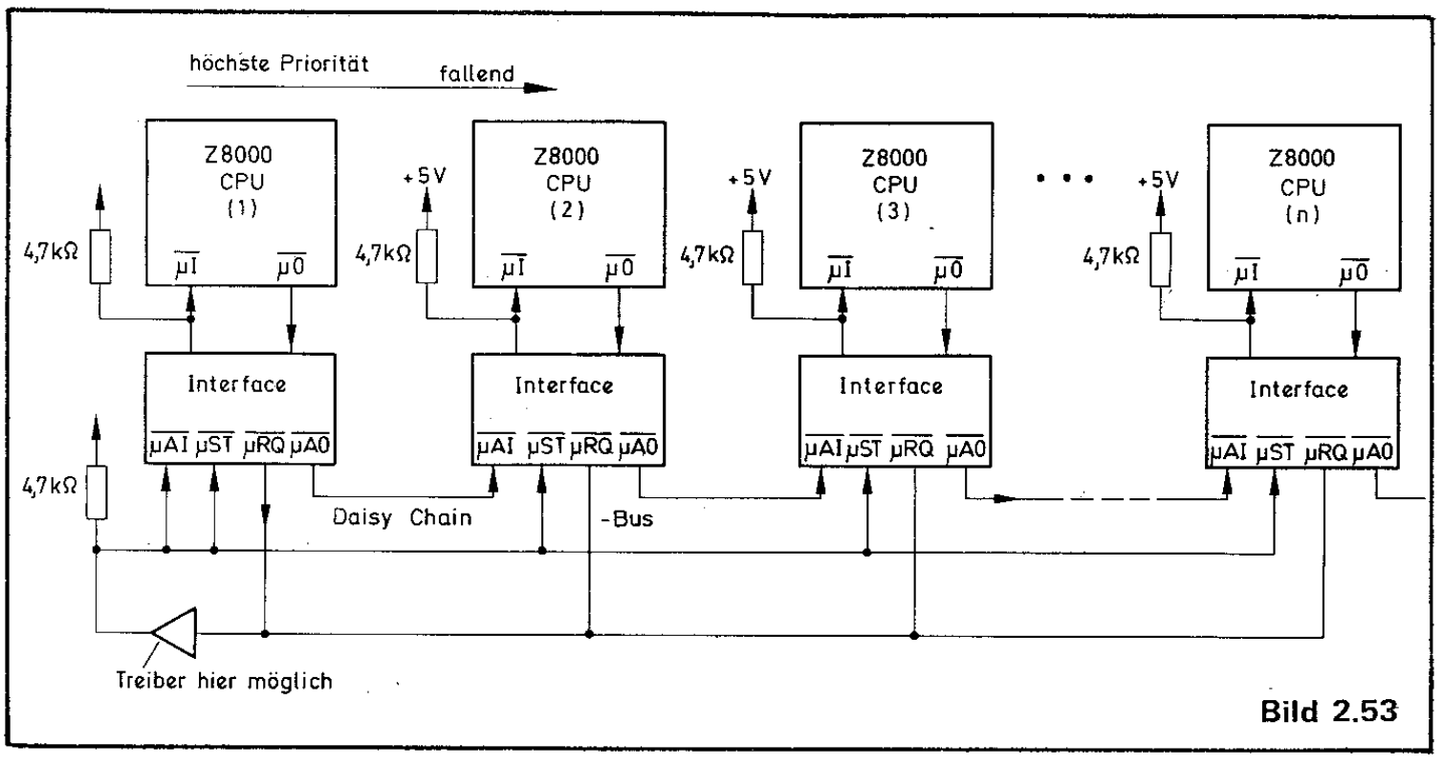


Bild 2.53

stellung in **Bild 2.54** auf der Z-Ebene sehr ähnlich. Die Reihe der Soldaten sind Subprozessoren bzw. Subsysteme (Z 8002). Die Host-CPU (Anführer) sendet den Befehl MSET aus (gleichsam mit abzählen). Das System (1) wird als höchst priorisiertes in der Kette zuerst aktiv. Die akustische Kennung "eins" wäre damit vergleichbar, daß dieses System mit Basisprogrammen aus dem main-frame-RAM initialisiert wird. Das kann extern geschehen (mit DMA) oder das System adressiert das RAM selbst. Ist das erste System fertig geladen, wird die CPU auf der Leitung $\overline{\mu A0}$ ein Lo zum nächsten $\overline{\mu AI}$ senden (gleichsam dem Vorgang des Kopf nach links drehens). Dieser Anerkennungs-Übergabemechanismus vollzieht sich System für System. Hat das letzte System in der Kette den Initialisierungsvorgang beendet, wird diese Sub-CPU den Ausgang $\overline{\mu O}$ inaktivieren und damit $\overline{\mu RQ}$ der Quelle ein Zeichen geben, daß die Initialisierung beendet ist. Nachdem $\overline{\mu RQ}$ des Systems (n) inaktiviert wurde, ist dies so oft mal geschehen, als Subsysteme in der Kette eingegliedert sind. Wird an $\overline{\mu AI}$ bzw. an $\overline{\mu I}$ angezeigt, daß die Initialisierung eines Systems beendet ist, wird auch $\overline{\mu RQ}$ eines Subsystems für einige Takte Hi sein und damit erkennbar und zählbar für die Quelle. Erkannt wird das durch die Abfrage von $\overline{\mu I}$ der Host-CPU,

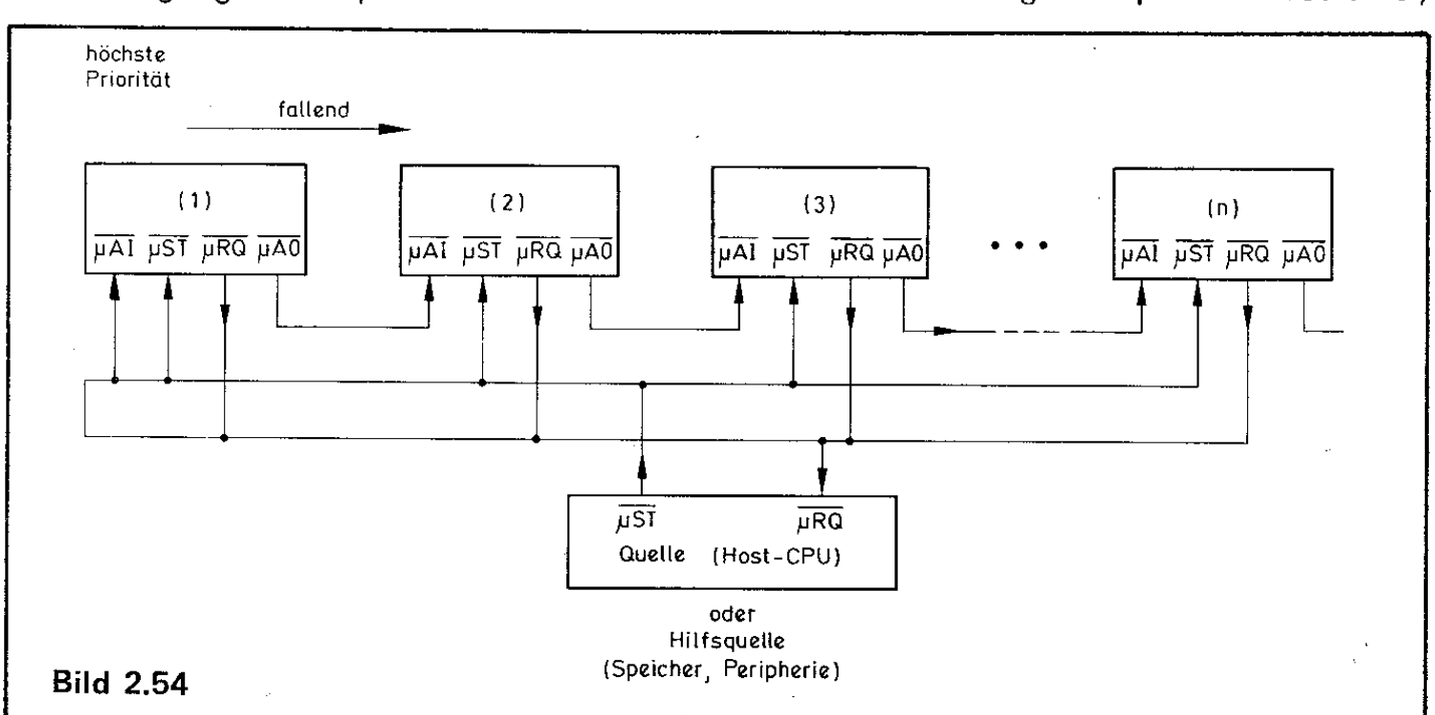


Bild 2.54

der mit $\overline{\mu RQ}$ identisch ist. Abgefragt wird mit dem Befehl MBIT. Die Abfrage wird ständig wiederholt, indem über JR (relativer Sprung) mit der Bedingung S-Flag gleich 1 (cc = 0101) MBIT aufgerufen wird. War S gleich 1, wird ein Register dekrementiert, der mit der Anzahl-1 der Subsysteme gesetzt war. Hat dieses Register den Wert Null erreicht, weiß die Host-CPU von der Beendigung der Initialisierung aller Subsysteme.

Für die Subsysteme gilt bei der Initialisierung die repetierende Anwendung des Befehls MREQ vor ihrem Initialisierungsvorgang. Ist sie beendet, wird die Sub-CPU mit dem Befehl MRES $\overline{\mu O}$ inaktivieren. Für die Subsysteme ist $\overline{\mu RQ}$ ein Ausgang (gebildet aus $\overline{\mu O}$). Die Host-CPU hat einen ständigen Überblick über den Initialisierungsverlauf bzw. -stand der Subsysteme.

Irgendwie ist dieser Vorgang schon faszinierend, obwohl er nicht besonders kompliziert ist. Vielleicht ist gerade das so faszinierend daran.

Das kurze Beispiel der Initialisierung von Subsystemen zeigt das Zusammenwirken von Software mit den beiden Multi-Micro-Anschlüssen des Z 8000. Wie Bild 2.54 zeigt, muß die Quelle nicht eine Host-CPU sein. Sie kann auch nur ein Schreib-/Lesespeicher sein (main frame-RAM). Eine übergeordnete Organisation des Multiprozessor-Systems würde demnach nicht existieren. Das stimmt jedoch nur zum Teil. In diesem Fall liegt die Priorität bei dem System (1) der Kette. Die CPU dieses Systems kann jederzeit eine Multibusverbindung anfordern, die sie auch zu allererst zugesprochen bekommt, wie bereits erwähnt wurde. Ihre Priorität bezüglich der Orga-

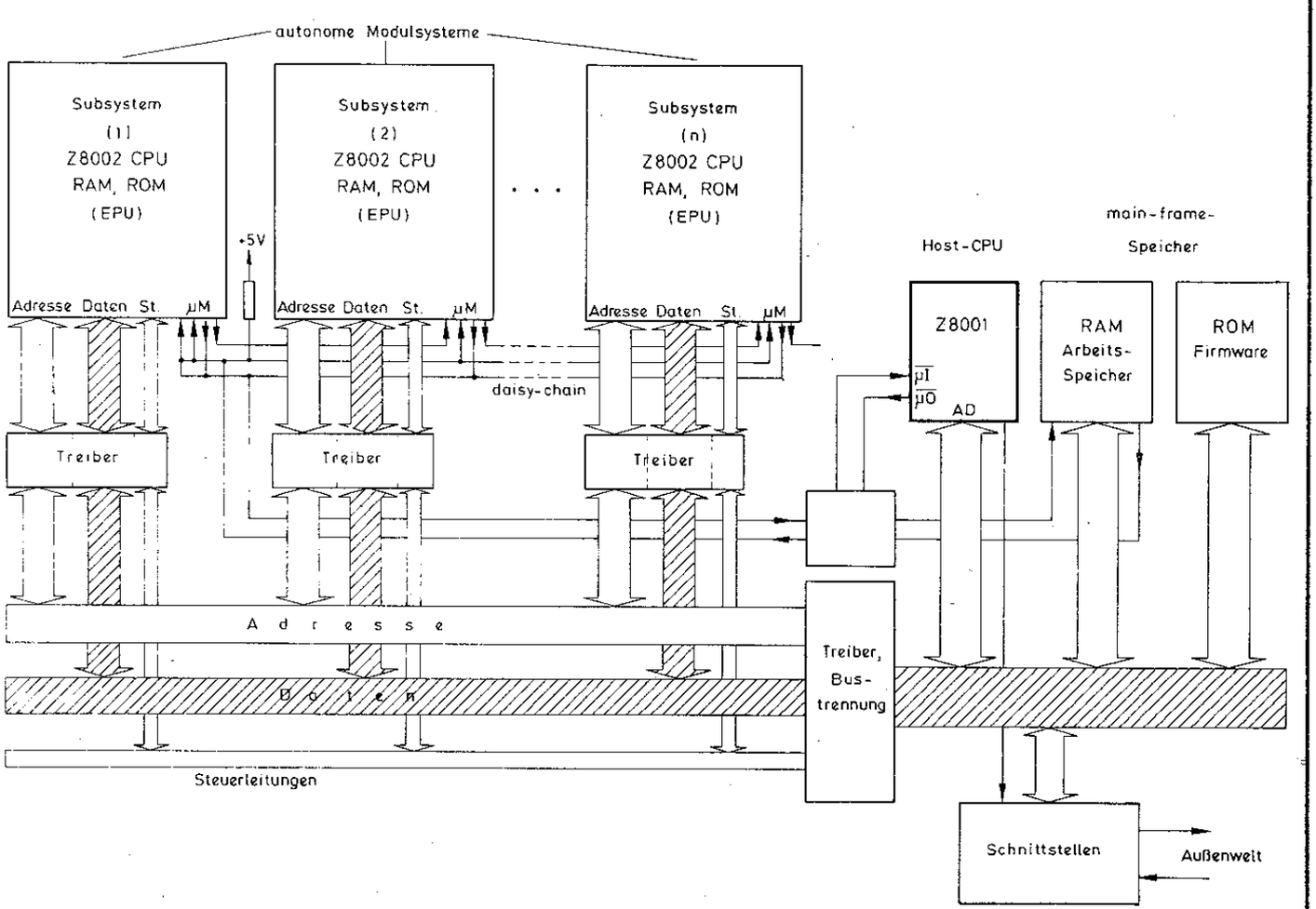


Bild 2.55

nisation des gesamten Systems liegt dann entsprechend schwerpunktmäßig auf dem Bereich des Betriebssystems. In diesem Bereich sind ohnehin in der Zukunft noch einige Aufgaben zu erfüllen.

Die Quelle kann sowohl ein Speichersystem und ein vielfältiges Schnittstellensystem gemeinsam sein. Wäre beispielsweise eine Host-CPU vorhanden, könnte diese den main-frame-RAM und die Schnittstellen betreuen, die Initialisierung der Subsysteme steuern, Programmergebnisse zusammenstellen und als Hauptaufgabe als Dialog-System für den Anwender gelten. Die Host-CPU hätte überwiegend Verwaltungsaufgaben. Die Subsysteme sind in einer solchen Konfiguration die eigentlichen Programmbearbeitungs-Organe. Die uns bisher bekannte timesharing-Betriebsart von Großrechnern würde sich nur noch auf den Dialog beziehen. Die Bearbeitung einzelner Anwenderprogramme wird in Echtzeit von Subsystemen ausgeführt. Für die Subsysteme können ebenso variable Ausbaustufen gelten. Einige davon können neben dem Arbeitsspeicher und der Sub-Firmware in PROMs aus einer Kombination einer Z 8002-CPU und einer Z 8000-EPU bestehen.

In **Bild 2.55** wird schematisch ein Multiprozessor-System mit Host- und Subsystem-Hierarchie dargestellt. Solche Architekturen sind möglicherweise die künftigen Großrechner aus 16-bit-CPUs. Sie sind in jedem Fall Alternativen zu gegenwärtigen Großrechnern. Die Leistungsfähigkeit von Großrechnern aus Mikroprozessoren muß erst noch auf breiter Basis bewiesen werden. Daß man sich jedoch sehr viel davon versprechen kann, dürfte unbestritten sein.

2.11.1 MULTI-MIKRO-DAISY-CHAIN-BUSKONFIGURATION

In der bisherigen Beschreibung wurde der "daisy-chain"-Bus als Mittel zum Zweck dargestellt. Für ein besseres Verständnis soll dieser Bus noch etwas genauer betrachtet werden.

In **Bild 2.56** ist die Kette der Systeme mit einzelnen Pegeln auf den Leitungen gezeigt. In dieser Formation ist das System (2) aktiv. Unter Aktivität ist der Fall zu verstehen, daß nach Bild 2.55 nur das System (2) mit dem Host-System in Verbindung steht, d.h. die Interface- und Treiberlogik von System (2) ist auf den Haupt-Transferbus durchlässig. Der aktive Zustand von System (2) zeigt an den Anschlüssen $\overline{\mu AI}$, $\overline{\mu ST}$, $\overline{\mu RQ}$ und $\overline{\mu AO}$ die Zustände Lo, Lo, Lo und Hi an. Um dies verstehen zu können, müssen die Anschlüsse des Multi-Micro-Interface erklärt werden.

$\overline{\mu AI}$ (Annahme für eine Anforderung):
An diesem Eingang nimmt die Sub-CPU ein Signal entgegen, daß ihr anzeigt, ob eine Anforderung auf die Quelle von den anderen CPUs in der Kette akzeptiert wird. Für die CPU der höchsten Priorität ist diese Anfrage überflüssig, denn ein Lo an $\overline{\mu RQ}$ wird sie als erste an $\overline{\mu AI}$ empfangen. Trotzdem soll der Befehl MREQ angewendet werden.

$\overline{\mu ST}$ (Status der Kette):
Eine CPU der Kette muß irgendwie feststellen können, ob eine andere die Quelle momentan benützt. Wird sie benützt, muß der Status der Kette logischerweise auf Lo sein, weil irgendein $\overline{\mu RQ}$ eines Subsystems auf Lo ist. Hat die Benutzung ein Ende gefunden, wird der Status der Kette auf Hi springen. Somit ist die Quelle im Moment frei geworden und darf mit Erfolg angefordert werden.

$\overline{\mu RQ}$ (Anforderung auf die Quelle):
An diesem Ausgang sendet die CPU mit Lo eine Anforderung auf die Quelle aus. Lo bleibt gesetzt und ist somit anerkannt, wenn an $\overline{\mu AI}$ nach einer bestimmten Laufzeit die Quittung mit Lo erhalten wird.

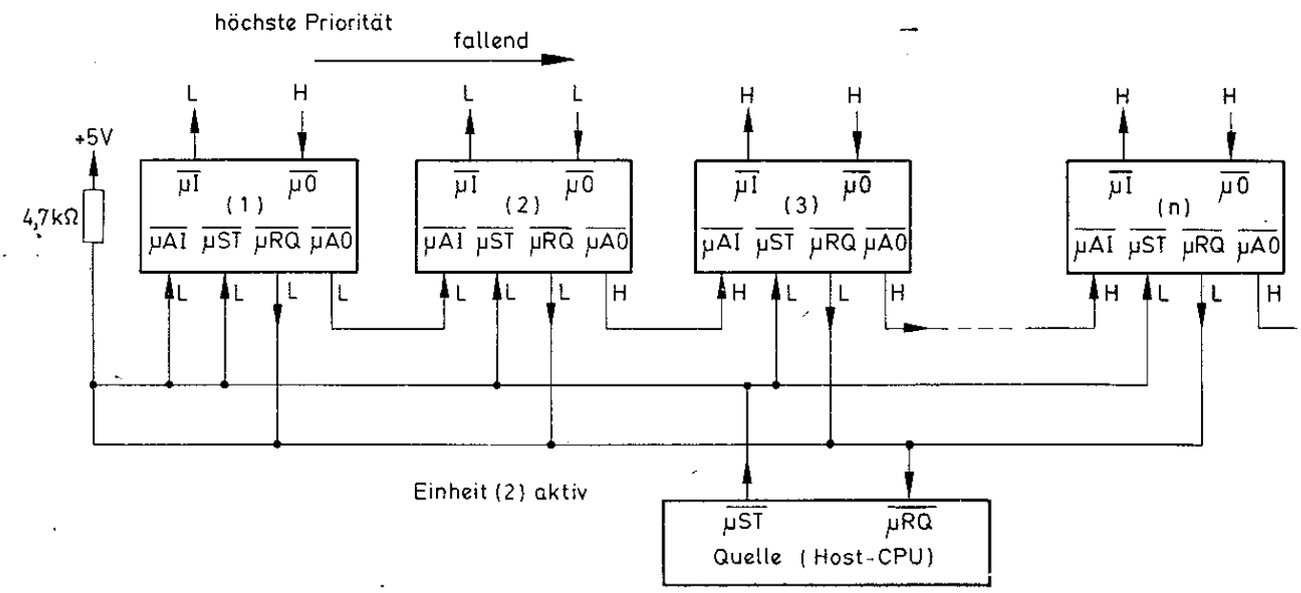


Bild 2.56

$\overline{\mu AO}$ (Anforderung weitergeben):
Ist $\overline{\mu RQ}$ gesetzt, wird grundsätzlich der Ausgang $\overline{\mu AO}$ immer Hi sein. Wird keine Anforderung ausgegeben (Hi), wird an $\overline{\mu AO}$ dann Lo ausgegeben, wenn auch an $\overline{\mu AI}$ ein Lo ankommt. Die Leitungen $\overline{\mu AI}$ - $\overline{\mu AO}$ bilden die Kettenschleife.

Sicherlich dürfte jetzt verstanden werden, daß die Status- und Anforderungsschleife den Widerstand auf +5 V hat. Damit wird eindeutig sichergestellt, daß bei einer Abfrage des $\overline{\mu ST}$-Eingangs immer dann eindeutig Hi anliegt, wenn keine CPU eine Anforderung an $\overline{\mu RQ}$ zur Quelle ausgegeben hat. Wäre $\overline{\mu ST}$ nicht sicher inaktiv, wenn es sein soll, würde keine Anforderung ausgegeben werden. Läßt man den Widerstand weg, könnte der Bus nicht oder zumindest nicht garantiert funktionieren.

In Bild 2.53 ist der "daisy-chain"-Bus mit dem separaten Interface für jedes Subsystem dargestellt. Wie geht's wohl da drinnen zu? **Bild 2.57** zeigt die Antwort. Es sind nur ein paar Gatter enthalten. Für das Beispiel Nr. 1 liegt vier mal Hi an, was für $\overline{\mu I}$ und $\overline{\mu O}$ gleich Hi bedeutet. Dieser Status gilt als absolut inaktiv. Keine CPU fordert die Quelle an. Im Beispiel Nr. 2 wird die Aktivität des Subsystems angedeutet. An $\overline{\mu AO}$ stellt man Hi fest, weil den niedriger priorisierten Subsystemen eine Anforderung auf die Quelle verwehrt werden soll. Sieht man sich noch einmal Bild 2.56 an, erkennt man den Status an den inaktiven Systemen. In Beispiel Nr. 2 war der Status für die aktive Einheit (2) angegeben. Folglich zeigt sich daraus der Status des höher priorisierten Systems (1) mit vier mal Lo an. An $\overline{\mu AO}$ liegt Lo, weil es die Anforderung von (2) durchgelassen hat. Bei System (3) ist $\overline{\mu AI}$ gleich Hi, weil es eine Anforderung nicht quittiert bekommen darf. Ebenso gilt das für alle niedriger priorisierten Systeme.

Konzentriert man sich nochmals auf die interne Logik des Interfaces, liegt in der Verbindung zwischen $\overline{\mu AI}$ und $\overline{\mu AO}$ erkennbar nur ein Inverter und ein NAND-Gatter. Die Laufzeit zwischen $\overline{\mu AI}$ und $\overline{\mu AO}$ beträgt somit etwa 20 ns, wenn $\overline{\mu O}$ inaktiv ist. Man stelle sich jetzt vor, alle Interfaces vor den Systemen wegzunehmen und die Anforderungs- und Anerkennungsschleife direkt an $\overline{\mu I}$ bzw. $\overline{\mu O}$ der CPUs zu verketten. Alle $\overline{\mu O}$s und $\overline{\mu I}$s sind verbunden und werden mit einem Widerstand im inaktiven Zustand auf Hi gezogen. Das bewirkt erst einmal eine Überlastung dieser CPU-Ein- und Ausgänge. Eine ordentliche Funktion ist hier ferner nicht möglich, denn eine Priorisierung der Kette ist nicht gegeben. Es entstünde genau die Verwirrung bei Zugriffen der Systeme, die man nicht

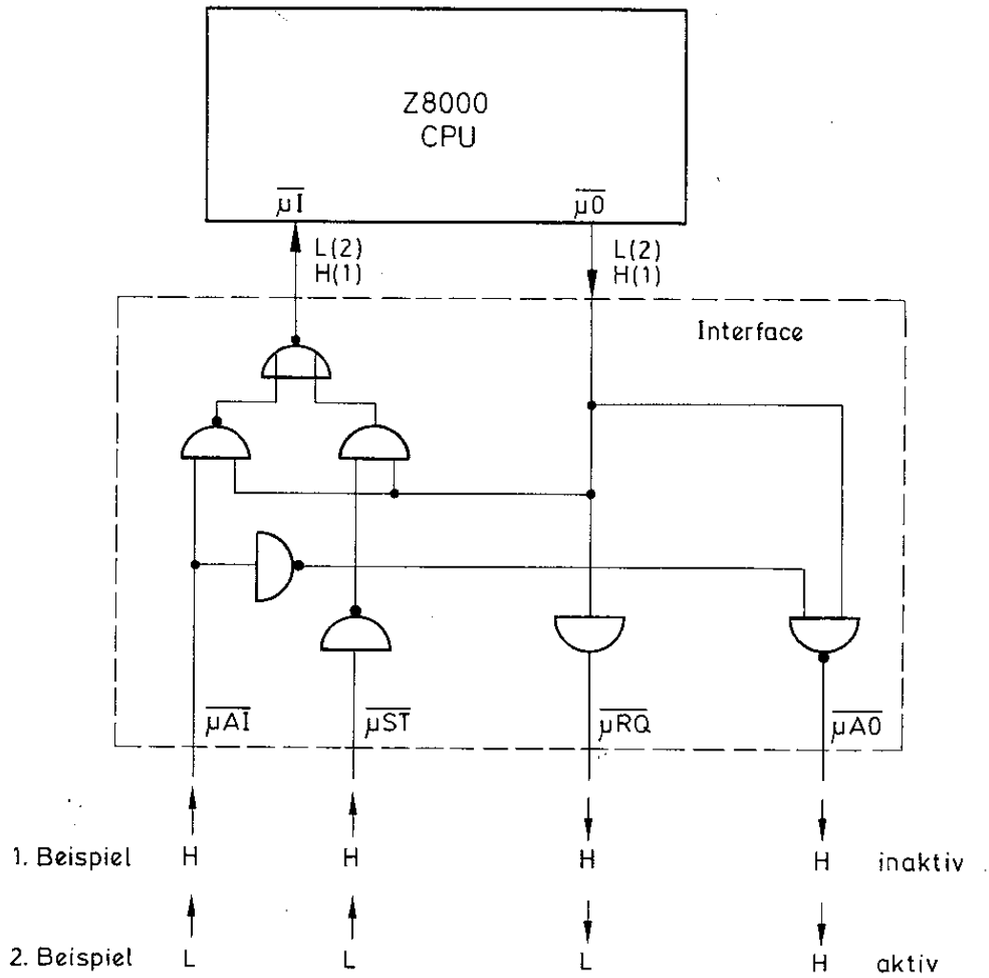


Bild 2.57

haben will. Die zweite Möglichkeit wäre, diese Kette sequentiell zu bilden. Von dem Ausgang $\overline{\mu O}$ einer CPU wird eine Verbindung zum Eingang $\overline{\mu I}$ der nächst niedrigeren CPU hergestellt. Die höchstpriorisierte CPU erhält an $\overline{\mu I}$ wieder den pull-up-Widerstand. Prinzipiell würde diese Version funktionieren. Abgesehen von der Belastung der CPU-Anschlüsse würde aber hier eine Verzögerung beim Durchschleifen einer Anforderung auftreten, die viel zu hoch wäre. Ein gering priorisiertes System z.B. an der 20. Stelle müßte 30 . . . 40 µs warten, bis es die Quittung für eine Anforderung erhält. Bei ungünstiger Programmierung kann es noch viel länger dauern. Beim Durchschleifen einer Anforderung des 20. Subsystems vergehen im Normalfall etwa 450 ns, wenn keine höher priorisierte Anforderung vorliegt.

Es gibt also keine Wahl, das beschriebene "daisy-chain"-Interface zu umgehen. Die Subsystemkette mit dem jeweiligen Interface in der Art nach Bild 2.53 ist der gangbare Weg.

In **Bild 2.58** ist eine praktische Realisierung dieses Interfaces dargestellt. Der vierfach 2-bit-Datenselektor und Multiplexer 74LS157 ist die einzigste 74er-Schaltung, die das Multi-Micro-Interface in einem Baustein ermöglicht, bräuchte man nicht noch den Treiber LS07 mit offenem Kollektor. Dieses Interface wäre technisch sicher leicht in einem 8poligen Mini-DIL-Gehäuse unterzubringen (ähnlich wie Operationsverstärker). Eine technische Realisierung wäre ein zu überdenkender Verbesserungsvorschlag. Da

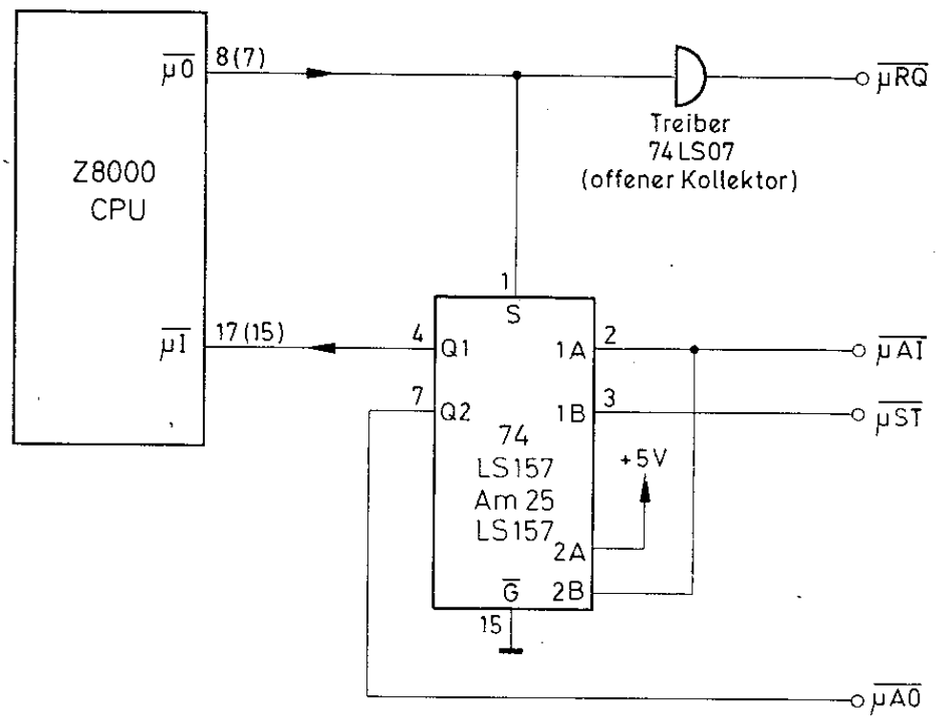


Bild 2.58

bis zum Erscheinen dieses Buches nichts in dieser Richtung bekannt wurde, muß man mit dem gegebenen auskommen. In **Bild 2.59** wird der Abschnitt des LS157 dargestellt, der für das Interface wichtig ist. Wird bei der Anwendung von MREQ zuerst $\overline{\mu ST}$ abgefragt, ist $\overline{\mu O}$ noch Hi, d.h. B1 swird an dem UND-Gatter durchlässig sein und an $\overline{\mu I}$ ein Hi liefern, wenn der Eingang $\overline{\mu ST}$ (B1) Lo geworden ist. Das bedeutet, daß keine andere Anforderung vorgelegen hat und demnach $\overline{\mu O}$ gleich Lo werden darf. Das bewirkt das Umschalten des Multiplexers auf die Eingänge A1 bzw. A2, wobei an A1 das Quittungssignal der Anforderung erwartet wird. Wenn $\overline{\mu O}$ gleich Lo geworden ist, wird Q2 bzw. $\overline{\mu AO}$ automatisch Hi.

Die Hardware des Multi-Micro-Busses ist damit abgeschlossen. Zuletzt wird nochmals auf die Softwarebehandlung eingegangen. Fordert ein Subsystem die Quelle an, muß die CPU ein Anforderungsprotokoll durchlaufen. Die Grundroutine ist im Flußdiagramm in **Bild 2.60** dargestellt. Die Anforderung auf die Quelle von einem Subsystem wird, wie mehrfach erwähnt, mit dem Befehl MREQ gemacht. Weil bei MREQ eine Schleife zu durchlaufen ist, die ein Register der CPU stetig dekrementiert und letztlich auf Null bringt, wird danach das Z-Flag gesetzt. Daher muß es zu Beginn der Routine auf Null gesetzt sein. Sodann wird $\overline{\mu I}$ das erste mal abgefragt, jedoch mit Hinblick auf $\overline{\mu ST}$ (Status). Wenn der

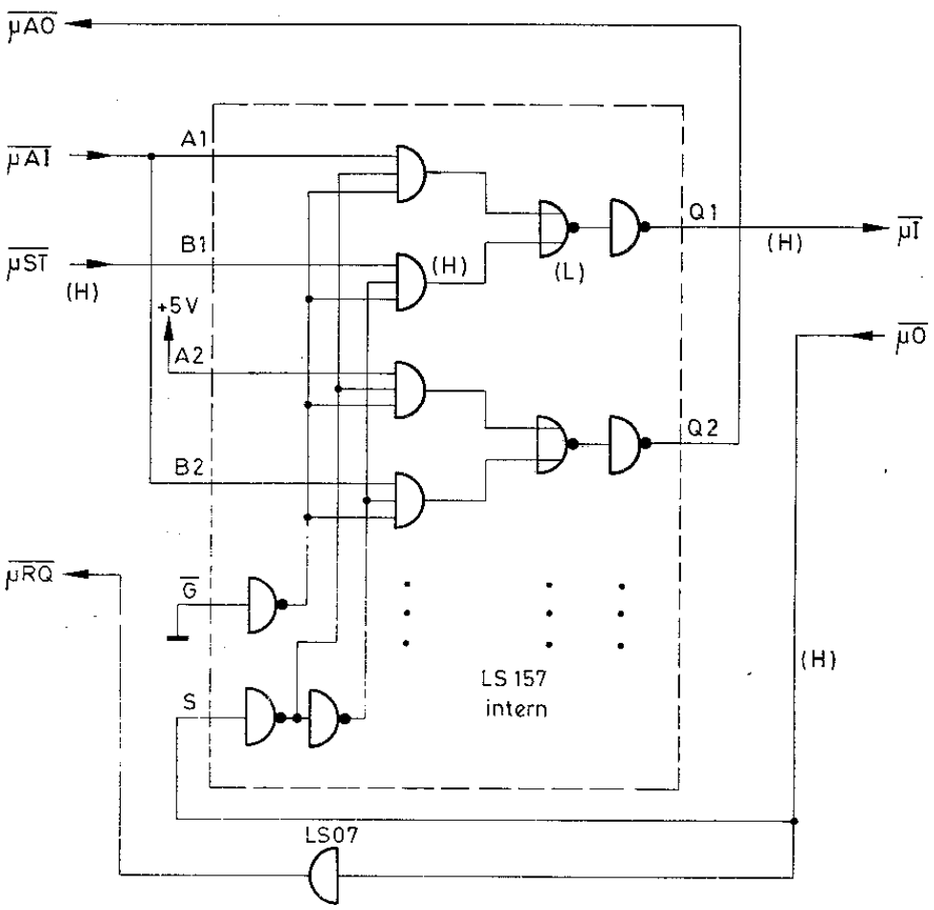


Bild 2.59

Eingang Lo ist, wird die Schleife begonnen. Andernfalls war die Quelle belegt und die Routine ist zu Ende. Wird in die Schleife gesprungen, dient diese dazu, an $\overline{\mu AI}$ eine Quittung für das Aktivieren von $\overline{\mu O}$ bzw. $\overline{\mu RQ}$ zu erhalten. Die Schleife muß zeitlich so dimensioniert sein, daß die Laufzeit bezüglich des Ranges des Subsystems in der Kette berücksichtigt ist. Man kann die Laufzeit des Quittungssignals des am niedrigsten priorisierten Subsystems der Kette ansetzen, plus einige Mikrosekunden. Bei 128 Subsystemen wären das etwa 8 µs. In dieser Zeit **muß** $\overline{\mu I}$ bzw. $\overline{\mu AI}$ Lo werden. Ist das nicht der Fall, ist die Schleife umsonst durchlaufen worden, weil ein höher priorisiertes Subsystem eine Anforderung generiert hat. Daraus folgt: $\overline{\mu I}$ muß immer wieder abgefragt werden. Irgendwann wird das Subsystem Erfolg haben und mit der Quelle eine Korrespondenz treiben können.

Es sei noch angemerkt, daß bei einer akzeptierten Anforderung kein System eine Unterbrechung der Korrespondenz bewirken kann, ausgenommen ein RESET der Sub-CPU. Ein RESET setzt $\overline{\mu O}$ unmittelbar auf Hi.

Multi - Mikro - Anforderungsprotokoll

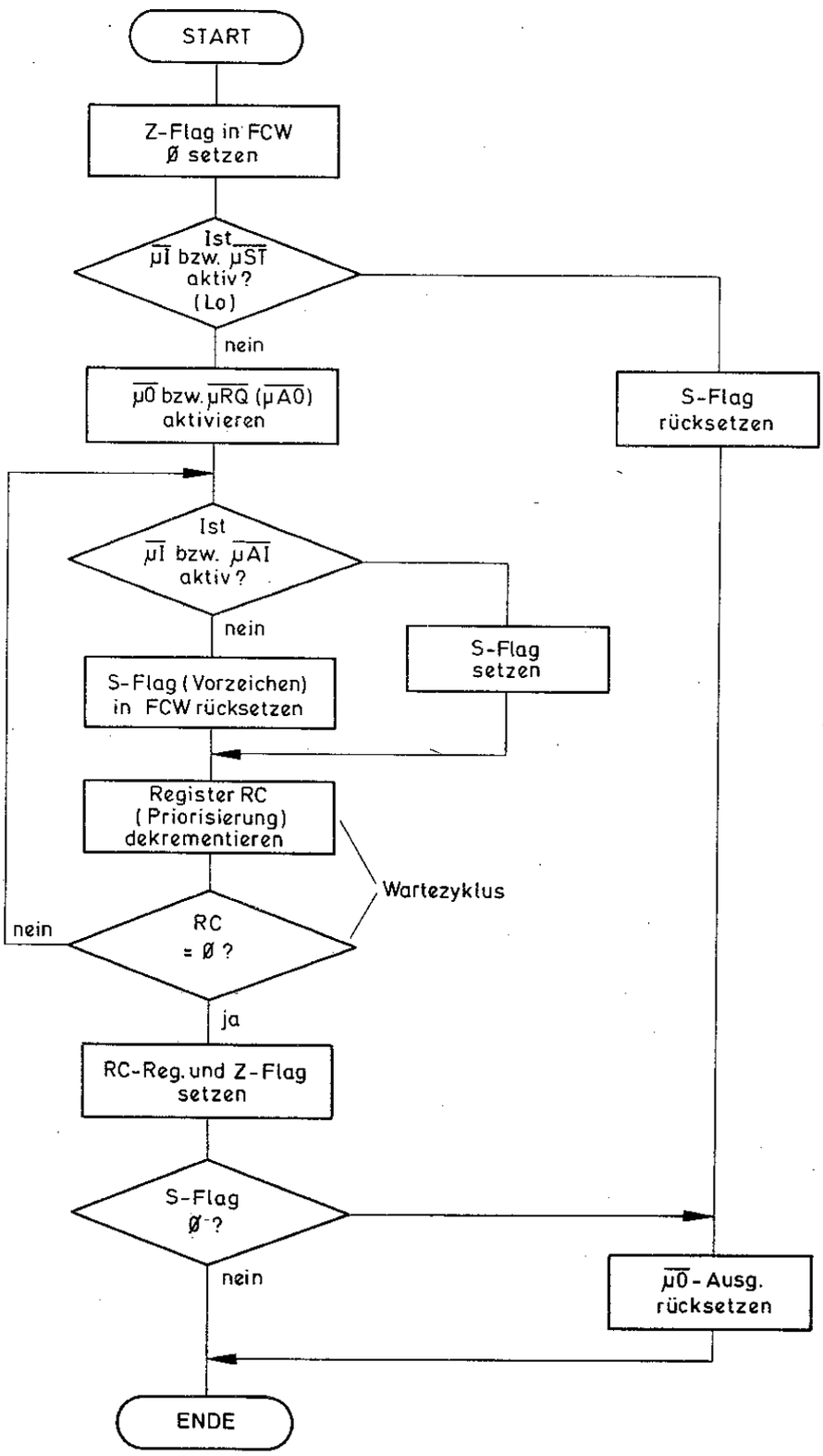


Bild 2.60

Adressierungsarten und Befehlssatz

3

Die in einem Computer errechneten **Adressen** sind ein Mittel zum Zweck. Unter dem Sammelbegriff Daten ist eine Vielzahl von Operanden zu verstehen, von denen einer oder zwei mit Hilfe eines Befehls verändert werden. Sind zwei Operanden in einer Befehlsausführung beteiligt, entsteht erfahrungsgemäß ein Dritter (Summe, Differenz, Produkt, Quotient). Werden sie verglichen, entsteht ebenfalls ein zusätzliches Argument, das aber meist unmittelbar als Entscheidungsquelle im Programm verwendet wird, ob gesprungen, normal weiter gearbeitet, angehalten oder sonstwie gehandelt wird. Wird nur ein Operand im Befehl bearbeitet, z.B. verschoben, entsteht ein zusätzlicher Operand, wobei der alte nicht unbedingt vernichtet werden muß.

All diese Operanden "fischt" sich die CPU aus der mehr oder weniger großen Masse im Speicher mit Hilfe einer ganz bestimmten Adresse heraus. Die neu entstandenen Operanden läßt sie bei Bedarf im Register oder schiebt sie unter irgendeiner anderen, aber trotzdem definierten Adresse in einen Speicherplatz.

Die Adresse definiert eindeutig den Ort eines Operanden im Datenspeicherfeld.

Dies ist gleichsam ein **Axiom** in der Datenverarbeitung. Jeder Rechner handelt danach. Generell zeigt die Adresse eine bestimmte Stelle eines Wortes an. In Abschnitt 2.8 konnte man sehr wohl feststellen, daß die CPU sowohl den Speicher als auch Schnittstellen adressiert. Das ist der vordergründige Unterschied einer Adresse, der mehr hardwaremäßig begründet ist. Speicher und Schnittstellen sind hardwaremäßig zwei verschiedene Objekte. Softwaremäßig wird hier jedoch kaum ein Unterschied gemacht. Über Schnittstellen erfolgt in gleicher Weise ein Transfer von Operanden wie mit dem Speicher. Im Speicher selbst unterscheidet man schon etwas genauer den Inhalt eines Speicherplatzes. Im Abschnitt 2.7 wurde der Status des Z 8000 besprochen. In der Tabelle der Statuswerte ist der Unterschied des jeweiligen Zugriffs auf den Speicher in genaue Beschreibungen gekennzeichnet. In dem Abschnitt wurde ferner noch darauf hingewiesen, woher die Adresse am Bus entnommen sein kann. Die Unterscheidung ist grundlegend. Die Adresse ist von der Art die gleiche, spricht jedoch von der Software her verschiedene Bereiche an.

Kommt die Adresse vom Programmzähler, wird im Speicher ein Programmbefehl angesprochen.
Kommt die Adresse von einem Register, wird ein Operand im Datenfeld angewählt.
Kommt die Adresse vom Stapelzeiger, wird eine Statusinformation angewählt.
Hinter einer Adresse verbirgt sich eine softwaremäßig zu unterscheidende Information. Man unterscheidet zwischen Programmcode-Wörtern, Daten-Operanden (im Speicher oder einer Schnittstelle) und Statuswörtern. Statuswörter benennen entweder Programmzählerwerte oder Flag- und Steuerworte. Obwohl letztere **keine Operanden** sind, könnte man diese Informationen noch unter den Oberbegriff Daten einreihen. Eine 16-bit-Offsetadresse ist eine beliebige Binärkombination. Durch die gleichzeitige Ausgabe des Status bekommt diese Dualzahl gleichsam einen Index mit hinzu. Die feine Unterscheidung über den Status bietet die Aufgliederung von Speicherblöcken in die drei Felder Programmcode, Daten und Stapel. Da dies sowohl für den Normal- als auch für den System-Modus gilt, ergeben sich daraus

2 mal 3 Felder, wie es in **Bild 3.1** dargestellt ist. Die Daten werden, wie schon erwähnt, in Speicher- und E/A-Operanden getrennt, das ebenfalls aus dem Status hervorgeht.

Diese Adresse mit Index bietet aus diesem Grund über das 64 KByte-Feld hinaus ein größeres absolutes Adreßfeld. Für jeden Index stehen 64 KByte zur Verfügung. Rechnet man zusammen: Code, Daten (Speicher, E/A) und Stapel. 4 Felder ergeben für den Z 8002 ein Adreßfeld von 256 KByte. Nimmt man eine Unterscheidung nach Bild 3.1 vor, sind es 6 Felder und somit 384 KByte. Zählt man noch ein E/A-Feld hinzu (E/A nur im System-Modus), ergibt sich für den Z 8002 ein Adreßfeld von immerhin **448 KByte.**

Für den Z 8001 ist die Adressierbarkeit in noch bedeutend größere Dimensionen angelegt. Eine Adresse ohne Index ist 23 bit lang. Daraus ergibt sich die Feldgröße von 8 MByte. Legt man alle 7 Felder in einer Zahl zusammen, bewegt man sich in astronomischen Bereichen.

2 mal 3 Grundfelder (Code, Daten, Stack) plus ein E/A-Feld ergeben einen adressierbaren Raum von 56 Megabyte für den Z 8001.

Diese Zahl ist in der Tat so gewaltig, daß sie einer Bemerkung bedarf. Das Stapelfeld enthält nur Rücksprung-Adressen und dazugehörige FCW-Werte für die jeweiligen Unterprogramme. Mit dem PUSH-Befehl sind wohl auch Daten in den Stapel zu schieben und mit POP wieder zu holen, doch den Stapel als Datenfeld zu bezeichnen ist nicht ganz richtig. Es stehen hierfür zwar theoretisch 8 MByte zur Verfügung, doch, wer kann das füllen!!! Programmcode- und Datenfelder kann man füllen. Ebenso läßt sich ein E/A-Feld beträchtlich füllen, wenn man den Hardwaretrick eines Speicherfeldes für E/A benutzt. Die letzten Bemerkungen gelten ebenso zu den 448 KB des Z 8002 als hinreichend.

Die genannte Technik einer Statusanzeige für einen Transfer in einem Feld wird auch von anderen 16 Bit'tern praktiziert. Der 8086 unterscheidet an seinen drei Statusleitungen $\overline{S0}$ . . . $\overline{S2}$ zwischen Speicher- und E/A-Feld schreiben und lesen und einem Programmcode-Zugriff. Die Statusausgänge gelten jedoch nur im Maximum-Mode. Ist der 8086 in diesem Modus, kann der Status nur über den speziellen Bussteuerbaustein 8288 effektiv dekodiert werden. Im Minimum-Modus wird ein Status ausgegeben, der sehr an den 8085 erinnert. Die Leitungen ALE, DT/$\overline{R}$, $\overline{DEN}$ und M/$\overline{IO}$ sind mit den Z 8000-Leitungen $\overline{AS}$, $\overline{DS}$ und $\overline{MREQ}$ verwandt, wobei die Z-Leitungen in ihrer Verständlichkeit transparenter sind. Eine Statusdefinition mit ST 0 . . . ST 3 ist beim 8086 in der Form nicht gegeben.

Der Prozessorstatus FC 0 . . . FC 2 des 68000 zeigt hier schon diskretere Angaben. Er unterscheidet zwischen Programmcode- und Datenfeldern im Anwender-Modus und der übergeordneten (Supervisor)-Betriebsart.

Man kann im allgemeinen feststellen, daß die **Adresse** bei modernen 16-bit-Prozessoren nicht mehr für sich allein steht. Für den Fall des Z 8000-Status lassen sich daraus leicht dekodierbare, diskrete Signale erzeugen, die im allgemeinen als Inhibitsignale für Speicherfeldabgrenzungen zu benützen sind.

3.1 DIE ADRESSIERUNGSARTEN DER Z 8000-CPU

Die Adressierung bei einem Befehlsholzyklus erfolgt immer in einem absoluten 16-bit-Adreßwert, wobei damit der Offsetadreßwert noch von dem 7-bit-Segmentadreßteil des Z 8001 ergänzt wird. Bei jedem Holvorgang wird der Programmzähler inkrementiert. Bei einem Wortbefehl um 2, bei jedem Dop-

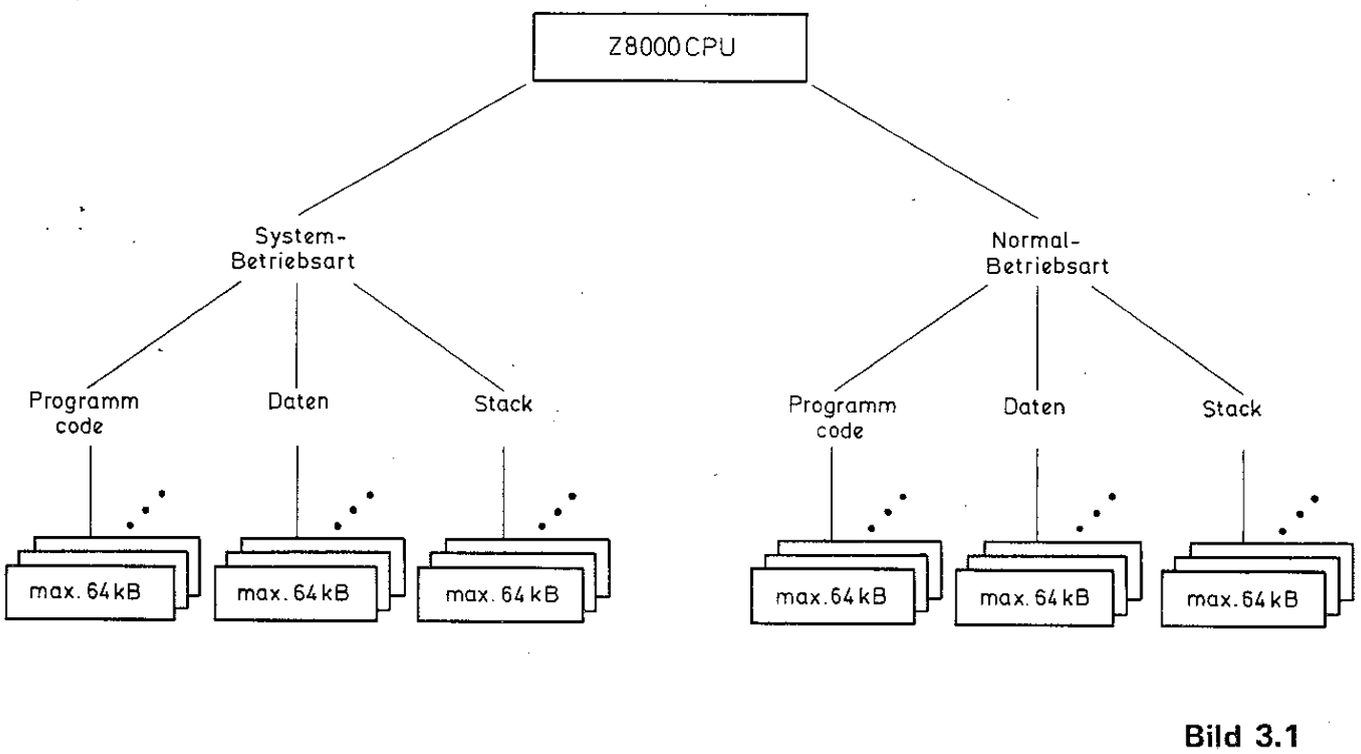


Bild 3.1

pelwortbefehl um vier Stellen usw. Der Programmzähler adressiert Befehlswörter. Adressiert er auch direkt Operanden?
Der Programmzähler holt auch direkt Operanden aus dem Programmspeicher, wenn in einem Befehl die Anweisung verknüpft ist, einen in ihm integrierten Operanden mitzuverarbeiten.

Damit ist der Kernpunkt dieses Abschnitts angesprochen. Um die Operanden dreht sich alles im Programm. Ein Operand wird auf verschiedene Arten adressiert. Wie die CPU einen Operanden zu adressieren hat, weiß sie nicht selbständig. Die Adressierung eines Operanden muß in jedem Befehl festgelegt werden. In Bild 2.2 wurde erstmals angedeutet, daß im Befehlswort an den Stellen 15 und 16 die Festlegung der Adressierungsart vorgenommen werden muß. Es gibt einige Z 8000-Befehle mit einem starren Muster (z.B. MRES und MSET), die keine Variation der Adressierungsart zulassen. Sie bilden aber eine wirkliche Ausnahme. In **Bild 3.2** ist das Befehlsformat noch einmal dargestellt, wie es bei der überwiegenden Zahl von Befehlen anzutreffen ist.

Das zu Beginn angesprochene Beispiel

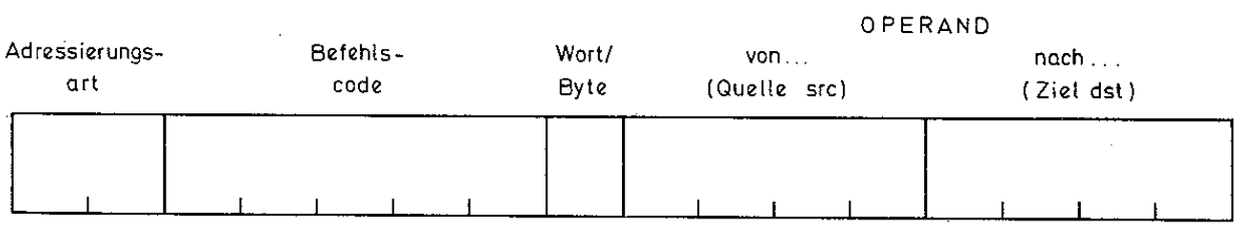


Bild 3.2

der unmittelbaren Adressierung wird als exemplarisch genauer ausgeführt. Wird der Operand unmittelbar adressiert (IM) ist in dem 2-bit-Feld der Adressierungsart der Wert 00 einzusetzen. Wird ein Befehl mit dieser Adressierungsart vom Steuerwerk der CPU dekodiert, weiß sie damit schon, daß sie an den nächsten beiden Speicherbytes einen Wortoperanden in das ausgewählte Register holen muß (vier Speicherbytes für Doppelwort-Operanden). Der Programmzähler wird aus diesem Grund nochmal um zwei bzw. vier Stellen inkrementiert. Nach der Ausführung des Befehls steht der Ergebnisoperand in dem dst- oder Zielfeld, das man ausgewählt hat (siehe Bild 3.2). An Hand des Subtrahier-Befehls wird gezeigt, wie man das praktisch ausführt. Nehmen wir an, die Zahl 16299 dez soll den Inhalt des Registers R 6 verringern. Der Assembler-Befehl lautet für diesen Fall:

SUB R6, #%3FAB

Im dst-Feld ist für R6 der Binärwert 0110 einzutragen. Das Quellfeld wird mit Nullen ausgefüllt. Der 5-bit-Befehlscode für SUB lautet 00001. Für Wort/Byte wird an dieser Stelle eine 1 eingesetzt, weil es eine Wortoperation ist. Für die Adressierungsart wird der Wert 00 eingesetzt, wie bereits angegeben wurde. Mit diesen Binärelementen kann man nun das Befehlswort SUB nach folgendem Bitmuster zusammensetzen:

Befehl (erstes Wort):
0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 — LSB

Der Operand 16299 lautet nach hexadezimaler Darstellung % 3FAB. In diesem Buch ist auch die Form 3FABH anzutreffen, jedoch nicht in einem Befehl. Der vom Programmzähler adressierte Operand lautet also:

Operand (zweites Wort):
0 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 — LSB

Will man den Z 8000 in hexadezimalen Werten programmieren, hat man einiges zu tun. Bei jedem Befehl muß man entscheiden, welche Adressierungsart man einsetzen will. Programmiert man in Hex, muß man die Stelle 15 und 16 des Befehlswortes beachten. Programmiert man im Assembler-Quellcode, muß man auf die Formulierung des Quelloperanden achten. Er steht hinter dem Komma der Zielregisterangabe.

Bis jetzt haben wir eine Adressierungsart des Z 8000 kennengelernt. Daß dieses Meisterstück wohl mehr als eine kennt, versteht sich von selbst. Er kennt 8 Adressierungsarten, die für eine Vielzahl der Befehle anwendbar sind:

- Operand steht im Register (Registeradressierung)
- Operand steht im Befehl) (unmittelbar)
- Operand steht mit Quelladresse im Register (indirekt)
- Operand steht mit Quelladresse im Befehl (direkt)
- Operand steht mit Basisadresse im Befehl und Distanzadresse in einem Register (indiziert)
- Operand steht mit Distanzadresse im Befehl und Basisadresse im Programmzähler (relativ)
- Operand steht mit Distanzadresse im Befehl und Basisadresse in einem Register (Basis)
- Operand steht mit Basis- und Distanzadresse in zwei unabhängigen Registern (Basis indiziert)

Wenn man es genau nimmt, wird ein Operand vom Programmzähler nur dann direkt im Codespeicher adressiert und geholt, wenn er im Befehl steht. Daß man diese Adressierungsart nicht all zu oft wählt, ist leicht einzusehen. Kommen in dem Programm entsprechende Variablen in größerer Zahl vor, müßten diese ständig im Codespeicher variiert werden.

Zum Zweck der besseren Trennung zwischen Code- und Datenspeicherfeld wird häufig die direkte und die indizierte Adressierungsart gewählt. Man kann mit ihrer Hilfe die Trennung sehr eindeutig

festlegen. Will man das Programm anwenden, kann man in einer separaten Initialisierungsroutine für die Ladung neuer Operanden (Variablen) in ein Datenspeicherfeld sorgen. Die unmittelbare Adressierungsart wendet man nur dann an, wenn es bestimmte Programmkonstante erfordern. Das Programm kann letztlich sehr allgemein formuliert werden. Sind die Adressen der Programmvariablen hintereinander angelegt, ist die Initialisierung per DMA machbar. Diese Methode wird oft in der Prozeßdatenverarbeitung angewendet.

Die komplizierteren Adressierungsarten sind die mit einer Basisadresse in einem Befehl oder einem Register. Die indizierte Adressierung gehört hier dazu. Die effektive Operandenadresse wird im Verlauf der Befehlsausführung berechnet. Die als absolut deklarierte Basisadresse steht in einem Befehl, zu der noch ein Vektor (besser ist jedoch ein Offsetwert) zur Berechnung der effektiven Operandenadresse hinzugezählt wird. Wie schon gesagt, die Basisadresse muß hier nicht nur im Befehl stehen. Steht sie beispielsweise im Programmzähler, ist damit die relative Adressierungsart angedeutet. Steht sie in einem beliebigen Register, ist die generelle Basis-Adressierungsart gewählt. Die Basis indizierte Adressierung gilt in diesem Rahmen als die komplizierteste, weil der Inhalt zweier Allzweckregister zur Berechnung der effektiven Operandenadresse herangezogen wird. Befehle dieser Adressierungsart sind durchweg Doppelwort-Befehle.

Geht man noch weiter ins Detail, unterscheidet man bei diesen höheren Adressierungsarten zwischen Programmzähler- und Register-Art. Die Basisadresse für einen Operanden steht entweder im Programmzähler oder in besagtem Allzweckregister. Wendet man den Befehl JR (Sprung relativ) an, wird der Programmzähler um eine bestimmte Distanz (im Befehl festzulegen) erhöht oder erniedrigt. Diese Adreßberechnung erfolgt ausschließlich programmbedingt, d.h. sowohl die Sprungweite (plus Richtung) als auch die Voraussetzung (Bedingung) für den Sprung muß im Befehl festgehalten sein. Man gibt entweder den relativen Sprung als Zahl an oder man setzt hinter der Bedingung ein sogenanntes Label. Das ist nichts anderes als die Befehlsbezeichnung, auf den der Programmzähler nach zutreffender Bedingung zeigen soll. Das Programmbeispiel in Abschnitt 2.5 zeigt einen relativen Sprung. Entweder gibt man PC-8 oder den Befehl SLLL an. Der Programmzähler muß absolut um 8 Stellen zurückgesetzt werden. Ein weiterer typischer Vertreter der relativen Adressierung ist neben JR noch CALR (Aufruf einer Programmroutine nach einem relativen Sprung).

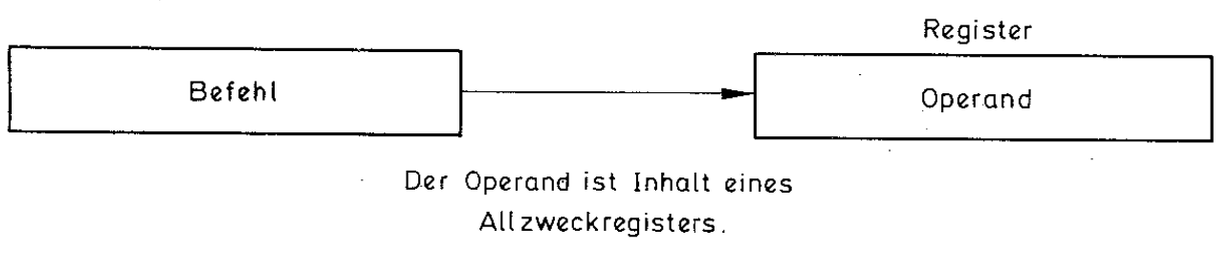
Bei der Ausführung des PUSH-Befehls werden zwei unterschiedliche Adressierungsarten gleichzeitig angewandt. Ein als Stapelzeiger benütztes Register enthält die Basisadresse des zu verschiebenden Operanden. Bevor der Operand in den Stapel gelegt wird (besser auf den Stapel), ist die Basisadresse (effektive Stapelspeicheradresse) um zwei Stellen erniedrigt worden. Wo kommt der Operand her? Erskann sowohl direkt aus einem Register stammen oder er muß aus dem Datenspeicher geholt werden. Die Ausführung des PUSH-Befehls wird immer in der Basisadressierung ausgeführt, wobei der Quelloperand in fünf anderen Adressierungsarten angewählt wird. Die Erniedrigung des Stapelzeigers um 2 Stellen (4 Stellen bei Doppelwort) repräsentiert die Basisadressierung. Da nun dieser Befehl so verdrahtet ist, daß immer diese Werte von der Stapelzeigeradresse abgezogen werden, spricht man vereinfachend von der indirekten Adressierung. Der Operand wird beispielsweise indirekt im Datenspeicher angewählt und in genannter Weise auf den Stapel gelegt.

Man spricht von Quell- und Zieladressierung bei einem Befehl. Überwiegend ist es der Fall, daß der Operand direkt, indirekt oder indiziert im Speicher angewählt wird, um anschließend im Register

verarbeitet zu werden. Die Zieladressierung ist somit überwiegend die Registeradressierung.

Im weiteren Verlauf dieses Abschnitts werden die Adressierungsarten des Z 8000 genauer vorgestellt.

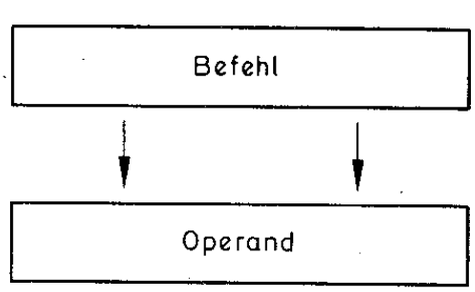
Registeradressierung (R):
Diese Adressierungsart ist die einfachste, direkte Art, die es bei Mikroprozessoren und auch beim Z 8000 gibt. Im Befehlswort ist das 4-bit-Feld "dst" (Ziel) enthalten, in dem das Ergebnis der Operation stehen soll. Im 4-bit-Quellfeld "src" wird der Registeroperand angegeben, der im dst mit verknüpft werden soll. **Bild 3.3** zeigt den Zusammenhang. Welche Art von Allzweckregister verwendet wird (Byte-, Wort-, Doppelwortregister), ist im Befehl impliziert. Befehle dieser Art sind am kürzesten und werden am schnellsten ausgeführt.



Der Operand ist Inhalt eines Allzweckregisters.

Bild 3.3

Unmittelbare Adressierung (IM):
Die unmittelbare Adressierung wurde in ihrer Art bereits vorgestellt. **Bild 3.4** zeigt das wesentliche dieser Adressierungsart. Der Befehl selbst beinhaltet den Operanden, der im Programm verwendet werden soll. Befehle mit dieser Adressierung werden zur Initialisierung von Registern verwendet. Man kann beispielsweise einen Indexwert in irgendein Register laden, der als Anfangswert für einen Zählvorgang dient. Dieser Index folgt unmittelbar an der nächsten Speicherstelle (in diesem Fall) nach dem Ladebefehl. Ferner lassen sich mit IM Testprogramme aufbauen, die gewissermaßen fertig sind. Indem Testdaten/-operanden bereits enthalten sind, werden Programme dieser Art nur im Code-Speicher aufgerufen.



Der Operand ist im Befehl enthalten.

Bild 3.4

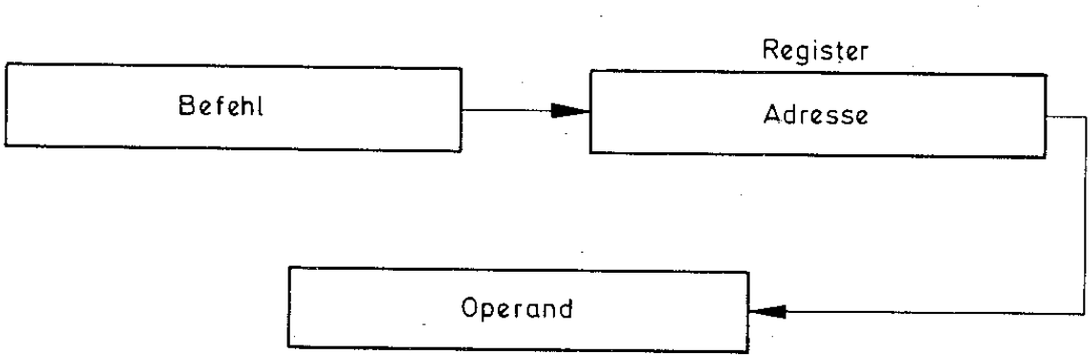
Indirekte Adressierung (IR):
Die indirekte Adressierung wird häufig verwendet, weil die effektiven Adreßwerte ein Datenfeld spezifizieren. Die Quellangabe im Befehlsformat deutet auf ein Allzweckregister hin (R 1 . . .

R 15), das diese Adresse eines Operanden enthält. **Bild 3.5** zeigt das Prinzip dieser Adressierungsart. Ein mit einer Adresse geladenes Register dient als Zeiger. Er muß einen absoluten 16-bit-Wert enthalten. Für den Z 8001 im segmentierten Modus sind zwei Wortregister (Offset, Segment) notwendig. Es kann jedes Allzweckregister, **ausgenommen R 0 bzw. RR 0,** dazu verwendet werden. Die Bezeichnung einer effektiven Adresse in einem Zeigerregister wird durch das Eingrenzen des src-Feldes mit Klammern (<>) gekennzeichnet. Im Assembler-Quellcode verwendet man ein spezielles Klammerzeichen (@ = Klammeraffe) für ein solches Zeigerregister. Als Beispiel sei dies an folgendem Befehl gezeigt:

SUB R0, @ RR 12.

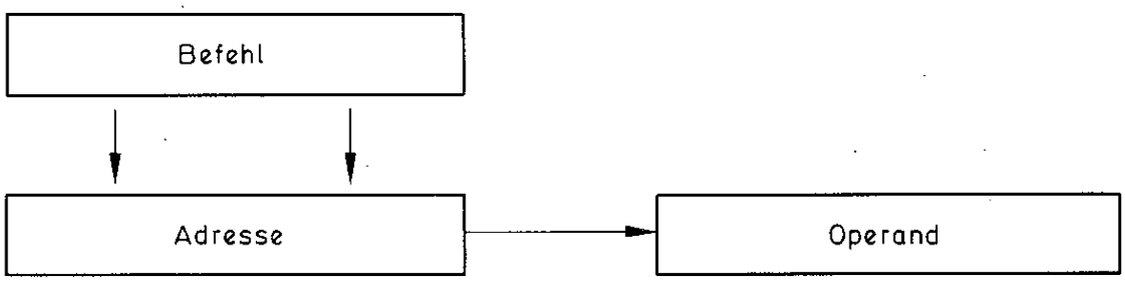
Das bedeutet, daß im Register RR12 die Segment- und Offsetadresse des Operanden steht, der von R 0 abgezogen werden soll.
IR ist deshalb sehr wichtig, weil die Adresse des Operanden eine Funktion der Zeigeradresse ist. Wird die Zeigeradresse geändert, wird folglich ein anderer Operand verwendet. Betrachtet man den Offsetteil als Index, kann man nach entsprechendem Inkrementieren oder Dekrementieren einfache Tabellenrechnungen anstellen. IR wird ferner bei E/A-Operationen benutzt, um die Adresse einer Schnittstelle ständig parat zu haben.

Direkte Adressierung (DA):
Diese Adressierungsart hat große Ähnlichkeit mit der direkten Adressierung und ist dementsprechend bedeutsam. **Bild 3.6** zeigt den Zusammenhang. Die Operandenadresse ist als quasi konstant anzusehen. Sie wird unmittelbar nach dem Befehlswort adressiert, steht also im Code-Speicher. Nach dem Holvorgang dieser Adresse steht sie kurzfristig im Register R 0, dessen vorhergehender Inhalt in der CPU zwischengespeichert wird, damit er nicht verloren geht. Nun gibt R 0 diese Operandenadresse aus und holt damit den Operanden aus dem Datenspeicher. Entweder wird er in ein Zwischenspeicherregister vor der ALU (wenn eine arithmetische Operation auszuführen ist) oder bei LD direkt in das angegebene Zielregister geladen. Ist der Operand geladen, wird in R 0 der alte Inhalt wieder zurückgeholt, womit wieder alles beim alten ist. Das src-Feld ist für einen Befehl in DA mit Nullen zu füllen. Damit wird auf die Hilfsquelle R 0 als Adreßregister hingedeutet. Ein Befehl in DA benötigt bei der Ausführung auch zwei Taktzyklen mehr als bei IR, weil sein Quelladreßregister unverändert bleibt, wogegen R 0 rekonstruiert werden muß.



Der Operand ist Inhalt einer Speicherstelle, dessen Adresse in einem Allzweckregister steht.

Bild 3.5



Der Operand ist Inhalt einer Adresse, die in dem Befehl mit inbegriffen ist.

Bild 3.6

Die direkte Adressierung wird bei den Befehlen JP und CALL benutzt. Hier ist das Ziel eines Adreß-Transfers der Programmzähler. Genauso wie bei der indirekten Adressierung wird mit der Adresse im Befehl auch eine Schnittstelle angesprochen, über die ein E/A-Transfer erfolgen soll. Im Feld für die Adressierungsart wird generell 01 eingesetzt. Der Assembler-Syntax läßt für DA zwei Arten der Beschreibung zu:

1. SUB R 10, % FFA0
2. SUB R 10, KONSTANTE

Im ersten Fall wird eine diskrete Quelladresse für den Subtrahend angegeben. Im zweiten Fall steht ein Label für einen Wert, der von R 10 abgezogen werden soll. Die Konstante ist gleichsam mit dieser Angabe örtlich bestimmt.

Auch bei dieser Adressierungsart ist für den Z 8001 zu unterscheiden, ob zwei oder ein 16-bit-Wort bei der Segmentadressierung benötigt werden. **Bild 3.7** erinnert hier noch einmal daran, speziell gedacht für JP und CALL.

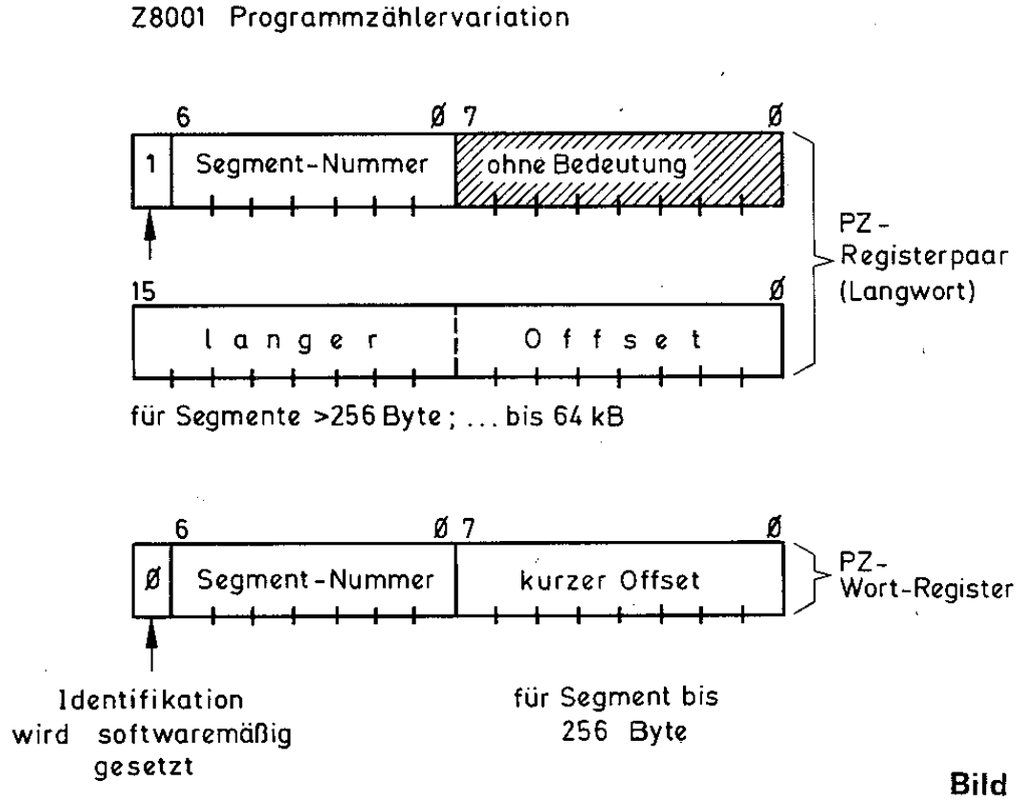


Bild 3.7

Indizierte Adressierung (X):
Bei der indizierten Adressierung findet eine Adreßberechnung statt. Um die effektive Adresse eines Operanden zu erlangen, muß die 16-bit-Baisadresse in einem Befehl mit der Distanzadresse in einem 16-bit-Quellregister (R1 . . . R15) addiert werden. **Bild 3.8** zeigt den Zusammenhang. Das Quellregister wird im src-Feld definiert. **Keinesfalls** darf **Register R0** als Register für die Distanzadresse benützt werden. Die indizierte Adressierung ist die zeitintensivste Adressierungsart des Z 8000. Im nicht segmentierten Modus ist ein Befehl durchweg zwei Worte lang, im segmentierten Modus mit langem Offset sind es drei Worte. Die indizierte Adressierung zeigt eine Ähnlichkeit mit der indirekten Adressierung auf. In beiden Fällen steht ein Adreßelement in einem Register. Für beide Fälle gilt, daß das Register als Index aufzufassen ist. Das Indexregister bei der indirekten Adressierung muß auf einen Anfangswert gesetzt werden. Bei der indizierten Adressierung kann man das machen, man muß es jedoch nicht. Als besonders positiv für den Programmierer zeigt sich bei der indirekten Adressierung die Hilfestellung Basisadresse. Die im Befehl integrierte Basisadresse kann als Anfangs- oder Endpunkt einer Datentabelle aufgefaßt werden. Die Summenbildung

Basis + Index = effektive Adresse
eines Tabellenelements

bietet die variable Eingliederung einer solchen Tabelle. Man setzt das gewählte Indexregister auf den relativen Endwert (bezogen auf die Basis) und bildet somit den Schleifenanfang. Nachdem die Manipulation des Adreßinhalts (letztes Element der Tabelle) beendet ist, wird das Indexregister dekrementiert und auf Null hin abgefragt. Ist das vorletzte Tabellenelement bearbeitet, wird wieder dekrementiert und abgefragt usw. Auf diese Weise wird jedes Tabellenelement leicht adressierbar. Es lassen sich Tabellen bis 64 KB Größe festlegen.

Relative Adressierung (RA):
Bei der relativen Adressierung steht die Basisadresse im Programmzähler und kann in einer gewissen Grenze verringert oder vergrößert werden. Die relative Adressierung ist eine Erweiterung der direkten Adressierung. Ein Adreßelement steht in beiden Fällen im Befehl. Neben dem relativen Aufruf- (CALR) und Sprungbefehl (JR) ist LDR (relatives Laden in Register) ein typischer Befehl dieser Art. CALR verfügt über eine Sprungweite von −2048 (+2047). JR verfügt über −128 (+127), wogegen LDR über die Sprungweite −32768 (+32767) verfügt. In **Bild 3.9** ist das Prinzip der relativen Adressierung dargestellt. Beim Ladebefehl LDR besteht das ganze zweite Befehlswort aus der Offset-Sprungadresse, aus der mit dem PC-Inhalt die effektive Operandenadresse berechnet wird. Eine Besonderheit zeigt der Assembler bei der Anwendung von LDR. Die Formulierung

LDR R5, $ + 16

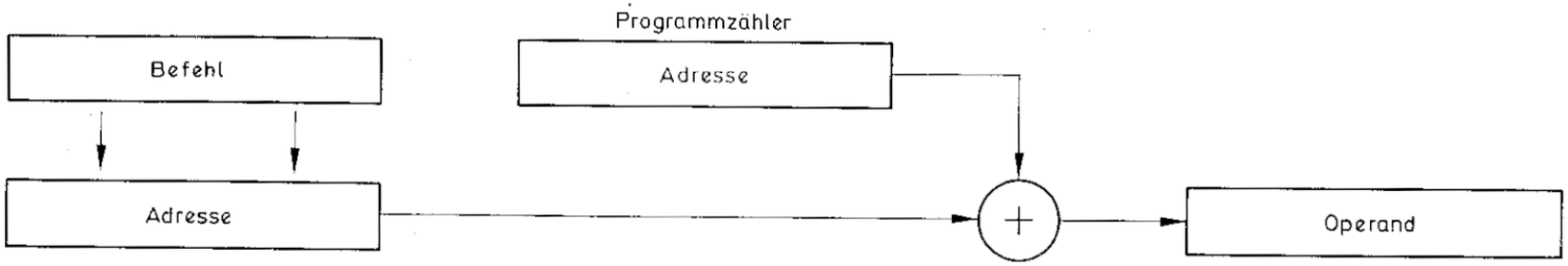
bedeutet, daß von Beginn des LDR-Befehls 16 Byte weitergesprungen wird, d.h. nach der Ausführung von LDR also nur 12 Byte. Die vier Byte von LDR werden automatisch abgezogen.

Konnte man bei der indizierten Adressierung noch ein im Rahmen des Datenformats mögliches Zweierkomplement einsetzen, muß das Sprungziel bei einem relativen Sprung genau bekannt sein. Der Offset stellt die Differenz zwischen dem Sprungziel und dem momentanen PC-Stand dar. Für JR liegt das Ziel im Bereich

$ − 128 ⩽ Sprungziel ⩽ $ + 127.

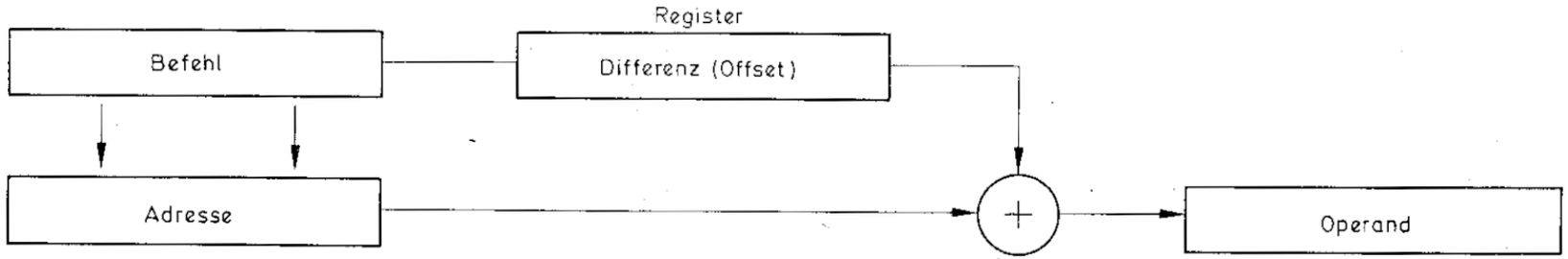
Für die anderen Befehle ist das entsprechend übertragbar.

Auch für die "relativen" Befehle gilt eine vereinfachte Beschreibung der Quelle, wie sie in der direkten Adressierung angewendet wird. Es erfordert keine Sprungweitenangabe in Zahlen mehr, sondern wird in Form eines Labels spezifiziert. Es kann demnach hinter dem Komma des Zielregisters bzw. der Be-



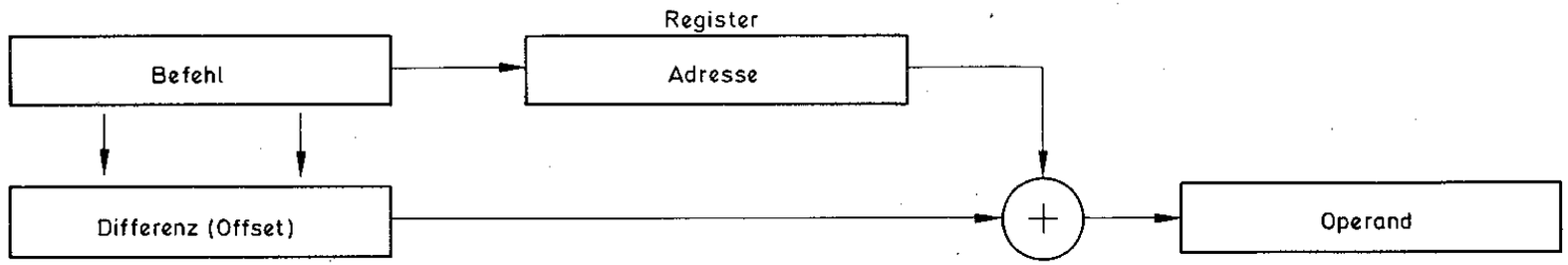
Der Operand ist der Inhalt einer Speicherstelle, die sich aus der Adresse im Programmzähler und dem Offset ergibt, der im Befehl inbegriffen ist.

Bild 3.9



Der Operand ist der Inhalt einer Speicherstelle, die sich aus der Adresse im Befehl plus der Differenzadresse (Offset, Distanz, -in einem Allzweck-Register abgelegt) ergibt.

Bild 3.8



Der Operand ist Inhalt einer Speicherstelle, die sich aus der Adresse in einem Allzweck-Register und dem Offset ergibt, der im Befehl inbegriffen ist.

Bild 3.10

dingung stehen.

```
JR    Carry, SORTIERE
LDR     R5, KONSTANTE
CALR      "Prozedure P"
```

Die relative Adressierung wird allgemein für verschiebbare (relocative) Programme benutzt, d.h. daß ein Block beliebiger Befehlscodes nicht von absoluten, physilatischen Speicherstellen abhängig ist. Das Programm muß selbstverständlich nach einer beliebigen Verschiebung noch funktionieren. Ein Vorteil ist, daß JR und CALR nur ein Wort benötigen. Man kann mit "relativen" Programmen Speicherplatz sparen. Es fragt sich, ob das heute noch ein stichhaltiges Argument sein kann.

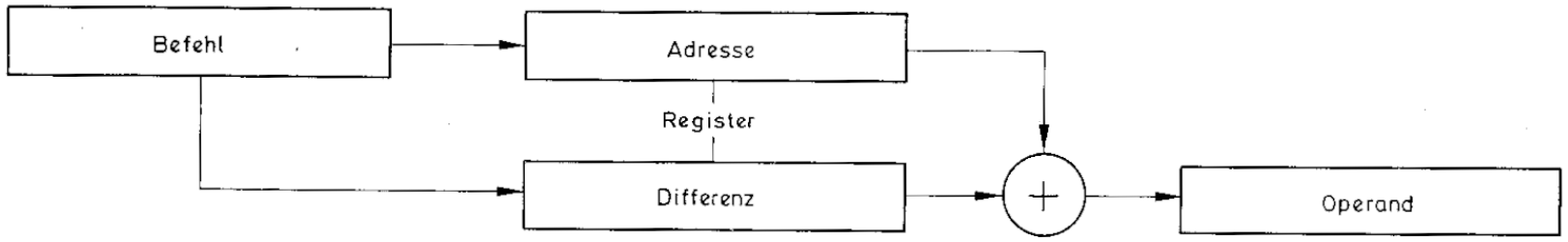
Basis-Adressierung (BA):

Die Basis-Adressierung ist die dritte Adressierungsart, bei der die Adresse eines Operanden berechnet wird. Sie ist sehr ähnlich zur indizierten Adressierung, die nur Basis und Distanz vertauscht hat. **Bild 3.10** zeigt das Prinzip. In jedem Allzweckregister darf die Basisadresse stehen (Segment, Offset), **ausgenommen R 0 bzw. RR 0.** Sowohl Offset als auch Distanz sind 16-bit-Adressen.

Im segmentierten Modus erlaubt die BA auch einen Zugriff auf Speicherstellen, dessen Segment im Befehl nicht angegeben sind. Die Beschreibung der Distanzadresse in einem Befehl wird mit dem Datenzeichen "#" angegeben, obwohl es kein Operand im eigentlichen Sinn ist. Um hier Mißverständnissen vorzubeugen wird ein Distanz-Adreßwert in Klammern gesetzt, z.B. (# % 2F3A) bedeutet den Hexadezimalwert = 12090 dez des Distanzwertes. Besonders Ladebefehle haben sich dieser Adressierungsart bemächtigt. Für andere Befehle wäre sie auch nicht sinnvoll.

Beispiel:

```
      LDL  RR 0, R 13 (#6721)
ist   LDL  RR 0, R 13 (#% 1A41)
ist   LDL  RR 0, R 13 (# Basis)
```



Der Operand ist Inhalt einer Speicherstelle, die sich aus der Adresse und dem Offset ergibt, die beide in Allzweck-Registern stehen.

In das Doppelregister RR 0 (R0 + R1) wird der Operand geladen, dessen nicht segmentierte Basisadresse im Wortregister R 13 steht.

Basis indizierte Adressierung (BX):

Die Basis-Indizierung ist eine weitere Verallgemeinerung der Basis-Adressierung. Zwei unabhängige Register dienen für den Basis- und Indexwert der Adresse. Die Addition beider ergibt die Operandenadresse im Speicher. **Bild 3.11** zeigt das Prinzip dieser Adressierungsart. Im segmentierten Modus steht die Basisadresse in einem Doppelregister (Segment, Offset). Die Distanz (Index) steht immer in einem Wortregister. Auch für diese Adressierungsart gilt wieder die Vereinbarung, daß sowohl Basis als auch Index (Distanz) **nicht in R 0 bzw. RR 0** stehen darf. Diese beiden Register sind nun mal keine Indexregister.

Die BX-Adressierung kann nur für Ladeoperationen benutzt werden. Ein Beispiel zeigt die syntaktische Behandlung der Basis-Indizierung.

LD R 0, R 13 (R 14)

bedeutet, daß in das Register R 0 der Operand geladen wird, der sich aus der Summe der Basisadresse in R 13 und des Index in R 14 ergibt. Im segmentierten Modus muß die Basisadresse in einem Doppelregister stehen. Der **Index** der Quelle wird **immer in Klammern** geschrieben. Es gibt überdies keine zwingenden Gründe, diese Adressierungsart bevorzugt anzuwenden, zumal sie zeitintensiv ist und bis zu drei Wortregister für die Adresse benötigt. Es kommt sehr auf die Anwendung an, Ladebefehle mit dieser Adressierung anzuwenden.

Befehle in komplizierteren Adressierungsarten anzuwenden, bringt mit der Zeit die Erfahrung mit sich. Übt man kräftig in der Assemblerprogrammierung, wird man langsam zum "Profi". Generell läßt sich jedoch sagen, daß es zeitlich in jedem Fall Verluste bringt, wenn mit LD in ein oder mehrere Register die Operanden erst geladen werden, um anschlies-

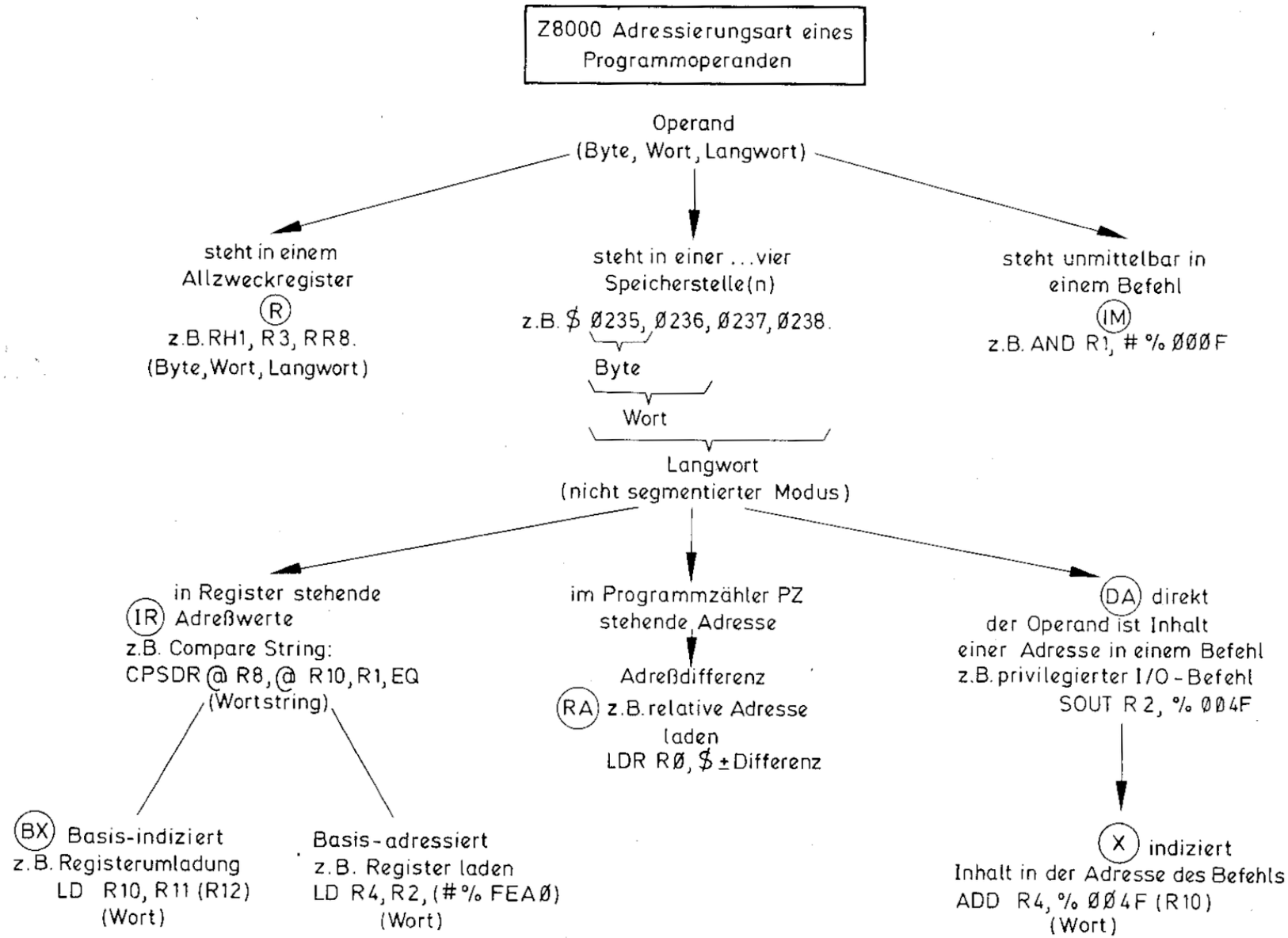

Z8000 Adressierungsart eines Programmoperanden
Operand
(Byte, Wort, Langwort)
steht in einem Allzweckregister
(R)
z.B. RH1, R3, RR8.
(Byte, Wort, Langwort)
steht in einer ...vier Speicherstelle(n)
z.B. $ Ø235, Ø236, Ø237, Ø238.
Byte
Wort
Langwort
steht unmittelbar in einem Befehl
(IM)
z.B. AND R1, #%ØØØF
(nicht segmentierter Modus)
in Register stehende
(IR) Adreßwerte
z.B. Compare String:
CPSDR @R8, @R10, R1, EQ
(Wortstring)
im Programmzähler PZ stehende Adresse
Adreßdifferenz
(RA) z.B. relative Adresse laden
LDR RØ, $ ± Differenz
(DA) direkt
der Operand ist Inhalt einer Adresse in einem Befehl
z.B. privilegierter I/O-Befehl
SOUT R2, %ØØ4F
(BX) Basis-indiziert
z.B. Registerumladung
LD R10, R11 (R12)
(Wort)
Basis-adressiert
z.B. Register laden
LD R4, R2, (#%FEAØ)
(Wort)
(X) indiziert
Inhalt in der Adresse des Befehls
ADD R4, %ØØ4F (R10)
(Wort)

Bild 3.12

send für eine Arithmetik die einfachste aller Adressierungsarten, die Registeradressierung, anwenden zu können. Wenn man so will, ist das Beispielprogramm in Kapitel 2.5 keineswegs optimal, weil dort dieser Fehler begangen wurde. Das Programm bearbeitet der Z 8000 noch schneller, wird er noch besser programmiert.

Ein einfaches Beispiel soll das hier verdeutlichen.

```
1  LD     R 2, % FFA0   9 T-Zyklen
2  ADD    R 3, R 2      4 T-Zyklen
```

ist prinzipiell gleich mit der Operation

```
1  ADD    R 3, % FFA0   9 T-Zyklen.
```

Im Register R 3 steht jeweils das gleiche Ergebnis, ob mit Übertrag oder nicht. Der ADD-Befehl ist zweifellos kürzer, wird er direkt im Register-Modus programmiert. Das Laden des Operanden dauert jedoch unweigerlich seine 9 Zyklen. Programmiert man die Addition mit der direkten Adressierungsart, ist man nicht nur schneller, sondern spart überdies noch eine Programmzeile. Oft liegt es daran, daß man sich nicht immer so traut, die Leistungsfähigkeit einer CPU über ihre Adressierungsarten voll auszunutzen.

Um abschließend in diesem Abschnitt die Adressierungsarten noch einmal zusammenfassend darzustellen, sind in **Bild 3.12** die Unterschiede des Ortes von Operanden aufgezeigt. Geht man von den drei Haupt-Datentypen aus, die der Z 8000 bietet, sind Operanden prinzipiell an drei Stellen aufzufinden: Register, Speicher, Befehl. Steht der Operand in einem Befehl, steht er wohl auch im Speicher, aber genaugenommen im Programmspeicher. Die Adressierungsarten entstanden rein logisch. Die des Z 8000 enthalten so viele Varianten, daß der Phantasie kaum noch Raum für weitere bleibt.

3.2 ALLGEMEINE BEFEHLSARCHITEKTUR

Der Befehlssatz des Z 8000 hat verschiedene Quellen. Er enthält viel bekanntes, eine Reihe Verbesserungen bekannter Befehlssätze etablierter Minicomputer und einige ganz neue Assembler-"Sprach"-Elemente. Insgesamt sind es 110 unterschiedliche Befehlstypen, die in 414 diskrete Befehle variiert werden können. Darüberhinaus gesellen sich einige "extended"-Befehle dazu, die die CPU zwar versteht, die sie aber nicht ausführen soll. Sie überläßt die Ausführung dem Baustein, für den sie geschaffen wurden, der Z 8000-EPU. Die CPU erzeugt einen "extended"-Trap, wenn ein Befehl dieser Art von ihr eingelesen wurde und das EPU-Bit im FCW auf Null gesetzt war. Sie bereitet sich andererseits bei der Ausführung von der EPU auf eine Kommunikation mit ihr vor. Hierauf wird in Kapitel 10 noch eingegangen.

Die Z 8000-Befehle lassen sich relativ übersichtlich in neun Anwendungsgruppen gliedern. Diese Gruppen beschreiben sich wie folgt:

- **Laden und ändern von Register- und Speicherinhalte**
  (einzelne Operanden, Daten-Strings, Adressen)
- **Arithmetische Operationen**
  (vier Grundrechnungsarten bis 32 bit Operandenlänge, auch mit Vorzeichen, Vergleich mit Register- oder Speicherinhalte, dekrementieren, inkrementieren, negieren)
- **Logische Operationen**
  (Boole'sche Funktionen AND, OR, Exklusiv OR, Komplement, Test).
- **Programmsteueroperationen**
  (direkte, relative Aufrufe, Schleifensteuerungen, einfach Branch, Sprünge <bedingt, relativ>, Rücksprünge)
- **Operationen für Einzelbit-Manipulationen**
  (Zero-Flag statisch und dynamisch abfragen, beliebiges Bit statisch und dy-

namisch setzen und rücksetzen, Vorzeichen-Flag abfragen und setzen)

- **Schiebe- und Rotationsoperationen** (in allen Registern rechts oder links, auch Doppelregister, über Carry, dynamisch, logisch, arithmetisch, rechts oder links)
- **Blocktransfer- und String-Manipulationen** (vergleichen und dekrementieren/inkrementieren, auch wiederholend, Einzeloperand oder Zeichenketten, laden und dekrementieren/inkrementieren, auch wiederholend, Speicher-Blockverschiebung, Blockverschiebung mit Test, auch wiederholend)
- **E/A-Schnittstellen-Operationen** (laden eines Byte-Operanden von Port in Register und umgekehrt, auch Wort, E/A-Operanden-Blocktransfer, dekrementierend/inkrementierend, auch wiederholend, Spezial E/A-Operationen mit MMU-Baustein, auch wiederholend)
- **CPU-Steueroperationen** (setzen und rücksetzen von Flags, Interrupt-Freigabe, CPU HALT, Programmstatus laden, lesen, Multiprozessor-Befehle, Operation Stop).

In diesen neun Gruppen sind einfache und komplizierte Befehle enthalten. Zu der Gruppe absolut komplizierter Befehle sind die Blocktransfer- und String-Manipulationen zuzuordnen. Die "Translate"-Befehle werden benutzt, um eine Tabelle von Byte-Strings von einem Code (z.B. ASCII) in einen anderen zu übersetzen. Ein Ziel-Indexregister adressiert das Tabellenfeld, das verändert werden soll. Die Basis zu diesem Index steht in einem Quelladreßregister, das auf die niedrigste Stelle der Tabelle von Übersetzungswerten zeigt. Der neue Indexwert wird aus der Addition des von dem Indexregister gerade adressierten Zielbyte in der Tabelle und der Quell- bzw. Basisadresse gebildet.

Index = Basis + momentan adressiertes Zielbyte in der Tabelle. Das Zielbyte ist mit mehreren Nullen an den höheren Stellen ausgebildet. Der neue Index adressiert einen 8-bit-Wert in der Tabelle, der nun neu geladen werden kann. Darauf hin wird der Zielindex dekrementiert oder inkrementiert. Die Operation kann auch wiederholend programmiert werden (TRDRB). Ein separates Wortregister dient als Stringelement-Zähler, d.h. Anzahl der Zeichen einer Tabelle.

Eine noch kompliziertere Form von String-Manipulationen zeigen die "Translate and Test"-Befehle. Hier wird mit zwei Quellregistern gearbeitet. Das erste dient zur Adressierung des Zielbyte und wird zugleich als Index benutzt. Das zweite zeigt auf den niedrigsten Wert der Tabelle.

Die Adreßberechnung für einen Tabellenwert ergibt sich aus der Summe des vom ersten Quellregister adressierten Tabellen-Zielbyte und dem Inhalt des zweiten Quellregisters, dessen 16-bit-Wert man folglich wieder als Basis auffassen kann. Wie für den Translate-Befehltyp zeichnet sich das Zielbyte mit Nullen in den höheren Stellen aus. Beim genannten, wie bei "Translate and Test" ist die Adreßaddition vorzeichenlos. Die Summenadresse zeigt auf einen 8-bit-Tabellenwert und lädt ihn in das Byteregister RH 1. War dieser Tabellenwert gleich Null, wird das Z-Flag gesetzt, wodurch sich der Test erklärt. Der Inhalt der vom zweiten Quellregister adressierten Speicherstelle bleibt unverändert. Das Index-Quellregister wird entweder dekrementiert (TRTDB) oder inkrementiert (TRTIB) und zeigt damit auf den nächsten Tabellenwert. Ein separates Wortregister dient wieder zur Zählung der Stringelemente. Es wird nach jeder Übersetzung dekrementiert.

Die Beschreibung des letzten Befehlstyps bezog sich nur auf eine Tabellenstelle. Auch dieser Befehl ist wiederholend programmierbar, und zwar den Index jeweils dekrementierend (TRTDRB) oder inkre-

mentierend (TRTIRB). Wie aus der jeweiligen Mnemonik am Schluß mit B festzustellen ist, kann man die repetierende Ausführung nur mit Bytes programmieren. Das besagte Stringzählerregister bestimmt die Anzahl der Tabellenwerte. Wurde es bis auf Null erniedrigt, ist der Befehl abgeschlossen.

Diese kapitalen Z 8000-Zeichenkettenbefehle suchen wirklich ihres gleichen. Befehle dieses Kalibers findet man bei anderen Minicomputern nicht, ganz zu schweigen bei 8-bit-CPUs. Befehle dieser Art benötigt auch ein "normaler" Assemblerprogrammierer nicht. Auf diese Befehle richten Compilerentwickler ihre Aufmerksamkeit. Nicht nur diese Befehle werden diese Leute unter die Lupe nehmen. Sie werden sich der nicht minder komplizierten "Compare-String"-Befehlen annehmen, die in gleicher Weise wiederholend programmierbar sind (CPSDRB, CPSIRB).

Der Z 8000 bietet in seinem Befehlssatz wirklich außerordentlich starke Varianten. Dies ist vielleicht die größte Stärke dieser CPU. Das muß ein noch so harter Konkurrent anerkennend unterstreichen. Echte Assembler-Profis wissen es zu schätzen, wenn eine CPU Befehle in der Art des Z 8000 besitzt.

Bis zu dieser Stelle des Buches wurden zwar viele Befehle erwähnt und teilweise besprochen. Ein genauer Überblick über die Assembler-Quellcodes wird jedoch erst jetzt gegeben. In der folgenden Liste werden die Z 8000-Befehle alphabetisch geordnet und mit einer Bemerkung versehen vorgestellt.

Tabelle der Z 8000-Befehlsätze in alphabetischer Reihenfolge

Alle Befehle mit "•" sind privilegierte Befehle, die **nur im System-Modus** ausgeführt werden können. Diese Vereinbarung gilt für beide CPU-Versionen.

ADCB
ADC (Addition mit vorhergehendem Übertrag, der dem Quelloperand hinzugezählt wird, B = Byteoperand)

ADDB
ADD
ADDL (Addition, mit Vorzeichen, Zweier-Komplement, B = Byte, L = Doppelwort)

ANDB
AND (logische UND-Funktion vom Quell- zum Zieloperand, B = Byte)

BITB
BIT (Abfrage eines bestimmten Bits in einem Wortregister, B = Byte)

CALL (retten des laufenden Programmzählerwertes in den Stapel, laden des neuen PC-Wertes)

CALR (retten des laufenden PC-Wertes in den Stapel, berechnen und laden des neuen PC-Wertes aus —PC—2xDistanz, entspricht: Zweier-Komplement von PC + Zweier-Komplement 2 x Distanz)

CLRB **CLR**	(löschen bzw. Null setzen eines Register- oder Speicherinhalts, B = Byte)
COMB **COM**	(Komplementbildung eines Register- oder Speicherinhalts, B = Byte)
COMFLG	(Komplementbildung der Flags C, Z, S, P oder V)
CPB **CP** **CPL**	(Vergleich eines Register- oder Speicherinhalts, Subtraktion, B = Byte, L = Doppelwort)
CPDB **CPD**	(Vergleichsfunktion Register oder Speicher mit dekrementieren eines Zählerregisters, B = Byte)
CPDRB **CPDR**	(Vergleichsfunktion mit dekrementieren, repetierend, B = Byte)
CPIB **CPI**	(Vergleichsfunktion Register oder Speicher mit inkrementieren eines Zählerregisters, B = Byte)
CPIRB **CPIR**	(Vergleichsfunktion mit inkrementieren, repetierend, B = Byte)
CPSDB **CPSD**	(Vergleichsfunktion zweier Zeichenketten im Speicher mit dekrementieren eines Zählerregisters, B = Byte)
CPSDRB **CPSDR**	(Vergleichsfunktion zweier Zeichenketten mit dekrementieren, repetierend, B = Byte)
CPSIB **CPSI**	(Vergleichsfunktion zweier Zeichenketten im Speicher mit inkrementieren eines Zählerregisters, B = Byte)
CPSIRB **CPSIR**	(Vergleichsfunktion zweier Zeichenketten mit inkrementieren, repetierend, B = Byte)
DAB	(Korrekturfunktion einer vorhergehenden Addition oder Subtraktion von Binär-BCD-Werte, nur für Byte-Arithmetik)
DBJNZ **DJNZ**	(Schleifensteuerung durch dekrementieren und abfragen, bei Null wird neuer PC geladen, Wortregister, B = Byteregister)
DECB **DEC**	(ein beliebiger Operand in Register oder Speicher wird um einen Wert zwischen 1 und 16 erniedrigt, B = Byte)
DI	(löschen bzw. Null setzen der beiden Freigabebits der maskierten Interrupts VI und NVI im Flag- und Steuerwortregister FCW)
DIV **DIVL**	(binäre Division, mit Vorzeichen, mit Ergebnis in Doppelwort- bzw. Vierfachwort-Register, L = 32-bit-Operand)
EI	(setzen der beiden Freigabebits der maskierten Interrupts VI bzw. NVI im FCW-Register)

EXB
EX
(Quell- und Zieloperand werden ausgetauscht, im Register oder Speicher, B = Byte)

EXTSB
EXTS
EXTSL
(das Vorzeichen-Bit der niedrigeren Hälfte eines Operanden wird auf die obere Hälfte des Operanden bezogen, B = Wortoperand, EXTS = Doppelwort-Operand, L = Vierfachwort)

- **HALT** (CPU-Operation ist ausgesetzt)

- **INB**
- **IN**
(mit Hilfe einer Eingangs-Portadresse wird ein Operand in ein Register geholt, B = Byteoperand)

INCB
INC
(ein beliebiger Operand in Register oder Speicher wird um einen Wert zwischen 1 und 16 erhöht, B = Byte)

- **INDB**
- **IND**
(laden eines Port-Operanden in einen Speicherplatz, mit dekrementieren eines Operandenzählers, B = Byteoperand)

- **INDRB**
- **INDR**
(laden mehrerer Port-Operanden (Datenblock) in Speicherplätze, mit repetierendem Dekrementieren eines Operandenzählers, B = Byteoperand)

- **INIB**
- **INI**
(laden eines Port-Operanden in einen Speicherplatz, mit inkrementieren eines Operandenzählers, B = Byteoperand)

- **INIRB**
- **INIR**
(laden mehrerer Port-Operanden (Datenblock) in Speicherplätze, mit repetierendem Inkrementieren eines Operandenzählers, B = Byteoperand)

- **IRET** (Rücksprung aus Interrupt-Serviceroutine, laden der Kennung, des FCW-Registers und zuletzt des PC-Wertes der letzten Programmzeile des vorhergehenden Programms vor der Interrupt-Anerkennung)

JP
(bedingter Sprung in eine Unterroutine, laden des neuen absoluten PC-Wertes)

JR
(relativer bedingter Sprung in eine Unterroutine, berechnen und laden des neuen PC-Wertes aus —PC + 2 x Distanz, entspricht: Zweier-Komplement PC + 2 x Distanz)

LDB
LD
LDL
(laden eines Operanden von Register oder Speicher in Register oder Speicher in allen typischen Adressierungsarten für Quelle und Ziel, B = Byteoperand, LD = Wort, L = Doppelwort)

LDA
(laden einer Adresse in Register)

LDAR
(berechnen und laden einer relativen Adresse in ein Allzweckregister aus dem momentanen PC-Wert nach LDAR plus Distanz)

LDCTLB
(nur Flags C, Z, S, P/V, D, H)

- **LDCTL** (laden eines Steuerwortes bzw. -Bytes in FCW, Refresh-Register, Programmstatus-Bereichszeiger 'SEG, OFF' und Stapelzeiger)

LDDB
LDD
(laden von Datenblöcken in Speicher)

LDDB
LDD
(laden von Datenblöcken in Speicher, mit dekrementieren eines Anzahl-Zählerregisters, B = Byteoperand)

LDDRB
LDDR
(laden von Datenblöcken in Speicher, mit repetierender Funktion durch dekrementieren eines Anzahl-Zählerregisters bis zum Wert Null, B = Byteoperand)

LDIB
LDI
(laden von Datenblöcken in Speicher, mit inkrementieren eines Anzahl-Zählerregisters, B = Byteoperand)

LDIRB
LDIR
(laden von Datenblöcken in Speicher, mit repetierender Funktion durch inkrementieren eines Anzahl-Zählerregisters bis zum Wert Null, B = Byteoperand)

LDK
(laden einer konstanten Zahl zwischen 0 und 15 in die niedrigsten 4 bit eines Wortregisters)

LDM
(repetierendes Transferieren des Inhalts im Allzweckregisterfeld in oder vom Speicher, Register 0 bis 15 wahlweise)

- **LDPS**
(laden Programmstatus, PC Offset, PC Segment, FCW)

LDRB
LDR
LDRL
(laden in Register vom Speicher oder umgekehrt mit einer relativen Sprungadreßangabe, effektive Operandenadresse aus Distanz plus momentanem PC-Inhalt, B = Byteoperand, L = Doppelwort)

- **MBIT**
(Abfrage des $\overline{\mu I}$-Anschlusses für ein Multiprozessorsystem, das S-Flag wird Null, wenn $\overline{\mu I}$ aktiv ist)

- **MREQ**
(Abfrage des $\overline{\mu I}$-Anschlusses, wenn aktiv → $\overline{\mu 0}$ nicht setzen, wenn nicht aktiv → $\overline{\mu 0}$ aktivieren, S-Flag abfragen)

- **MRES**
($\overline{\mu 0}$-Anschluß auf Null setzen, Hi)

- **MSET**
(μ-Anschluß auf Eins setzen, Lo)

MULT
MULTL
(binäre Multiplikation, mit Vorzeichen, mit Ergebnis in Doppelwort- bzw. Vierfachwort-Register, L = 32-bit-Operand)

NEGB
NEG
(Bildung des echten Komplements, Zweier-Komplement $x \rightarrow 2^n - x$, B = Byteoperand)

NOP
(keine CPU-Operation)

ORB
OR
(logische ODER-Verknüpfung des Quell- und Zieloperanden, B = Byteoperand)

- **OTDRB**
- **OTDR**
(Ausgabe eines Datenblocks in einen Ausgabeport, mit repetierendem Dekrementieren eines Operandenzählers, B = Byteoperand)

- **OTIRB**
- **OTIR**

(Ausgabe eines Datenblocks in ein Ausgabeport, mit repetierendem Inkrementieren eines Operandenzählers, B = Byteoperand)

- **OUTB**
- **OUT**

(mit einer Ausgangs-Portadresse wird ein Operand in einem Register zu einem Ausgangs-Port gegeben, B = Byteoperand)

- **OUTDB**
- **OUTD**

(Ausgabe eines Operanden vom Speicher an einen Port, mit dekrementieren eines Operandenzählers, B = Byteoperand)

- **OUTIB**
- **OUTI**

(Ausgabe eines Operanden vom Speicher an ein Port, mit inkrementieren eines Operandenzählers, B = Byteoperand)

POP
POPL

(zurückholen von Daten, Statuswerten oder Adressen aus dem Stapelspeicher in das Arbeitsregister oder Datenspeicher, L = Doppelwort-Operand)

PUSH
PUSHL

(inverser Befehl zu POP, verschieben von Daten, Statuswerten oder Adressen aus Arbeitsregistern oder Datenspeicher in den Stapelspeicher, L = Doppelwort-Operand)

RESB
RES

(setzt ein bestimmtes Bit in einem Operand, Wort oder Byte, auf Null, die Bitnummer ist eine unmittelbare Zahl)

RESFLG

(setzt ein oder eine Kombination der Flags C, Z, S, P oder V auf Null)

RET

(Rückkehr aus Unterroutine nach Bedingung, lädt aus dem Stapel den letzten Stand des PC vor dem Sprung ins UPRO wieder in den PC zurück)

RLB
RL

(schiebt Inhalt in Register an MSB links in Carry und am LSB wieder in das Register, B = Byteoperand)

RLCB
RLC

(schiebt Inhalt in Register an MSB links durch Carry und am LSB wieder in das Register, B = Byteoperand)

RLDB

(geschlossene Linksverschiebung eines BCD-Digits über drei Dezimalstellen am niedrigsten Digit wieder in das Register)

RRB
RR

(schiebt Inhalt in Register an LSB rechts in Carry und am MSB wieder in das Register, B = Byteoperand)

RRCB
RRC

(schiebt Inhalt in Register an LSB rechts durch Carry und am MSB wieder in das Register, B = Byteoperand)

RRDB

(geschlossene Rechtsverschiebung eines BCD-Digits über drei Dezimalstellen am höchsten Digit wieder in das Register)

SBCB
SBC

(Subtraktion mit vorhergehendem Übertrag, der dem Quelloperand hinzugezählt wird, B = Byteoperand)

SC

(ermöglicht einen Aufruf im Normal-Modus und hat das Ziel, die CPU in den System-Modus und/oder segmentierten Modus zu setzen)

SDAB
SDA
SDAL
(verschiebt Inhalt in Register arithmetisch nach links oder rechts, je nach Vorzeichen des Quelloperandes in einem anderen Register; die Zahl im Quelloperand bedingt die verschobenen Stellen, B = Byte 8 Stellen, Wort 16 Stellen, L = Doppelwort 32 Stellen)

SDLB
SDL
SDLL
(verschiebt Inhalt in Register logisch nach links oder rechts, je nach Vorzeichen des Quelloperanden in einem anderen Register; die Zahl im Quelloperand bedingt die verschobenen Stellen, B = Byte, Wort, L = Doppelwort)

SETB
SET
(setzt ein bestimmtes Bit in einem Operand, Wort oder Byte, auf Eins; die Bitnummer ist eine unmittelbare Zahl)

SETFLG
(setzt ein oder eine Kombination der Flags C, Z, S, P oder V auf Eins)

- **SINB**
- **SIN**

(einfacher Ladebefehl, MMU-Register zu CPU-Register, B = Byte)

- **SINDB**
- **SIND**

(Blockdaten-Transfer von MMU zu Datenspeicher, automatisches dekrementieren des Zieladreßregisters, mit dekrementieren eines Operanden-Zählerregisters, B = Byteoperand)

- **SINDRB**
- **SINDR**

(Blockdaten-Transfer von MMU zu Datenspeicher, repetierendes dekrementieren des Zieladreß- und Operanden-Zählerregisters, B = Byteoperand)

- **SINIB**
- **SINI**

(Blockdaten-Transfer von MMU zu Datenspeicher, automatisches inkrementieren des Zieladreßregisters, mit dekrementieren eines Operanden-Zählerregisters, B = Byteoperand)

- **SINIRB**
- **SINIR**

(Blockdaten-Transfer von MMU zu Datenspeicher, repetierendes inkrementieren der Zieladresse und dekrementieren des Operanden-Zählerregisters, B = Byteoperand)

SLAB
SLA
SLAL
(verschiebt Inhalt in Register arithmetisch nach links in Carry um die im src-Operand angegebene Nummer, B = Byte 0 . . . 8, Wort 0 . . . 16, L = Doppelwort 0 . . . 32)

SLLB
SLL
SLLL
(verschiebt Inhalt in Register logisch nach links in Carry um die im src-Operand angegebene Nummer, B = Byte 0 . . . 8, Wort 0 . . . 16, L = Doppelwort 0 . . . 32)

- **SOTDRB**
- **SOTDR**

(Blockdaten-Transfer von Datenspeicher zu MMU, repetierendes dekrementieren des Quelladreß-Registers und des Operanden-Zählerregisters, B = Byteoperand)

- **SOTIRB**
- **SOTIR**

(Blockdaten-Transfer von Datenspeicher zu MMU, repetierendes inkrementieren des Quelladreß-Registers und dekrementieren des Operanden-Zählerregisters, B = Byteoperand)

- **SOUTB**
- **SOUT**

(einfacher Ladebefehl, CPU-Register zu MMU-Register, B = Byte)

- **SOUTDB**
- **SOUTD**

(Blockdaten-Transfer von Datenspeicher zu MMU, automatisches dekrementieren des Quelladreßregisters, mit dekrementieren eines Operanden-Zählerregisters, B = Byteoperand)

- **SOUTIB**
- **SOUTI**

(Blockdaten-Transfer von Datenspeicher zu MMU, automatisches inkrementieren des Quelladreßregisters, mit dekrementieren eines Operanden-Zählerregisters, B = Byteoperand)

SRAB
SRA
SRAL

(verschiebt Inhalt in Register arithmetisch nach rechts in Carry um die im src-Operand angegebene Nummer, B = Byte 0 . . . 8, Wort 0 . . . 16, L = Doppelwort 0 . . . 32)

SRLB
SRL
SRLL

(verschiebt Inhalt in Register logisch nach rechts in Carry um die im src-Operand angegebene Nummer, B = Byte 0 . . . 8, Wort 0 . . . 16, L = Doppelwort 0 . . . 32)

SUBB
SUB
SUBL

(Subtraktion, mit Vorzeichen, Zweier-Komplement-Addition, B = Byte, L = Doppelwort)

TCCB
TCC

(bildet einen Operanden nach Booleschen Gesetzen als Folge einer logischen Operation des vorangegangenen Befehls; in dem 4 bit cc-Feld ist die abzufragende Bedingung enthalten, B = Byteoperand)

TESTS
TEST
TESTL

(testet einen Operanden nach einer logischen ODER-Verknüpfung, Test auf Inhalt NULL und gesetztem MSB, B = Byteoperand, L = Doppelwort)

TRDB

(Manipulation einer Zeichenketten-Tabelle, Stringanzahl-Register wird dekrementiert)

TRDRB

(Manipulation einer Zeichenketten-Tabelle, repetierende Bearbeitung, Stringanzahl-Register wird dekrementiert bis NULL)

TRIB

(Manipulation einer Zeichenketten-Tabelle, Stringanzahl-Register wird inkrementiert)

TRIRB

(Manipulation einer Zeichenketten-Tabelle, repetierende Bearbeitung, Index wird inkrementiert, Stringanzahl-Register wird dekrementiert bis NULL)

TRTDB

(Manipulation einer Zeichenketten-Tabelle mit einer Testfunktion, Stringanzahl-Register wird dekrementiert)

TRTDRB

(Manipulation einer Zeichenketten-Tabelle mit einer Testfunktion, repetierende Bearbeitung, Index wird dekrementiert, Stringanzahl-Register wird dekrementiert bis NULL)

TRTIB

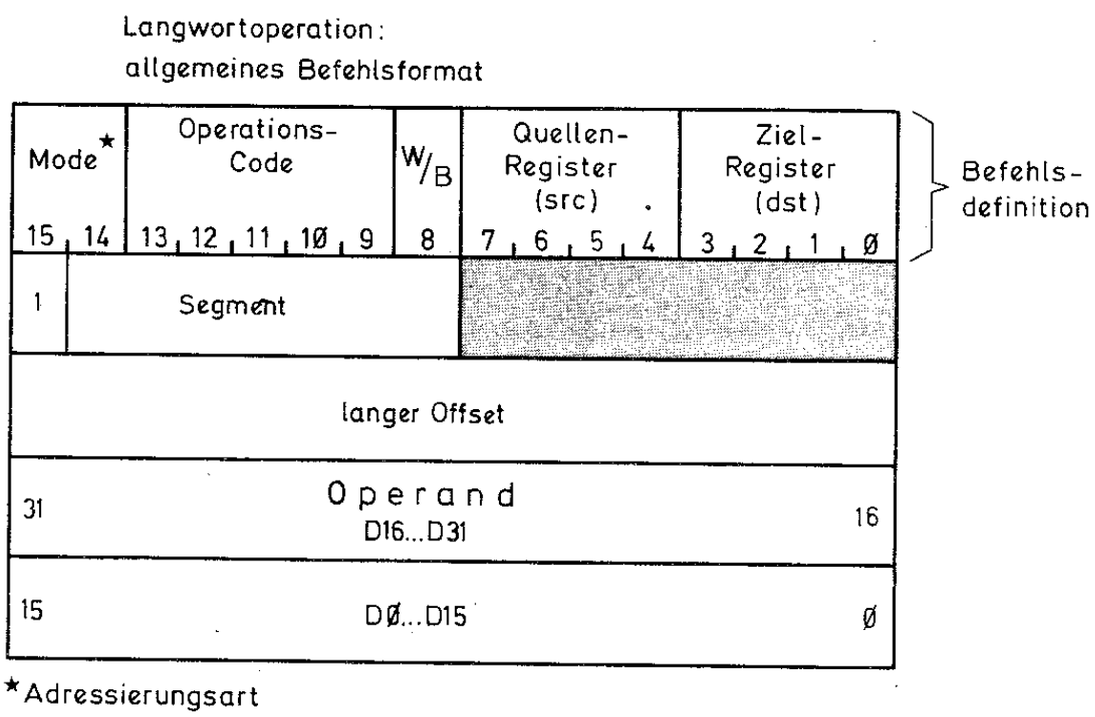
(Manipulation einer Zeichenketten-Tabelle mit einer Testfunktion, Stringanzahl-Register wird dekrementiert)

TRTIRB (Manipulation einer Zeichenketten-Tabelle mit einer Testfunktion, repetierende Bearbeitung, Index wird inkrementiert, Stringanzahl-Register wird dekrementiert bis Null)

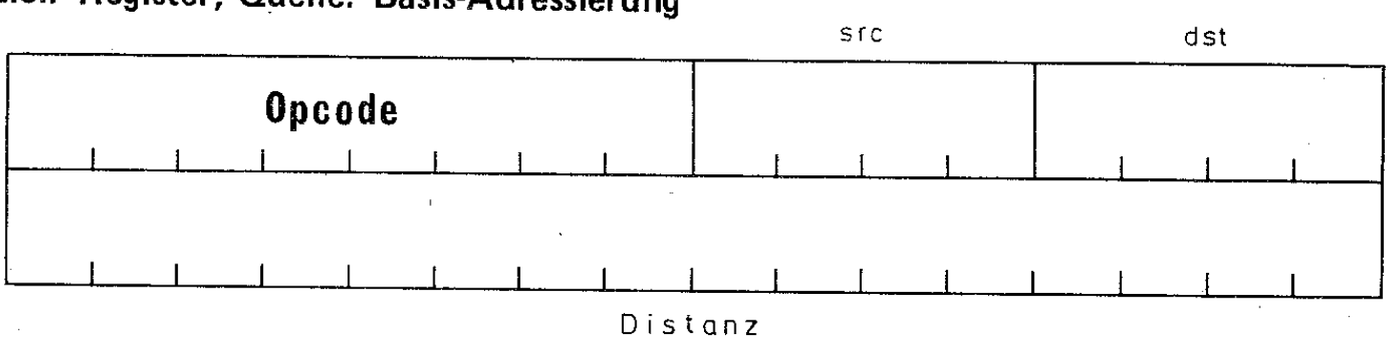
TSETB
TSET (Abfrage eines Operanden, ob das MSB gesetzt ist; war es gesetzt, alle Bits des Operanden werden auf Eins gesetzt, B = Byteoperand)

XORB
XOR (logische "exklusiv"-ODER-Verknüpfung des Quell- und Zieloperanden, B = Byteoperand)

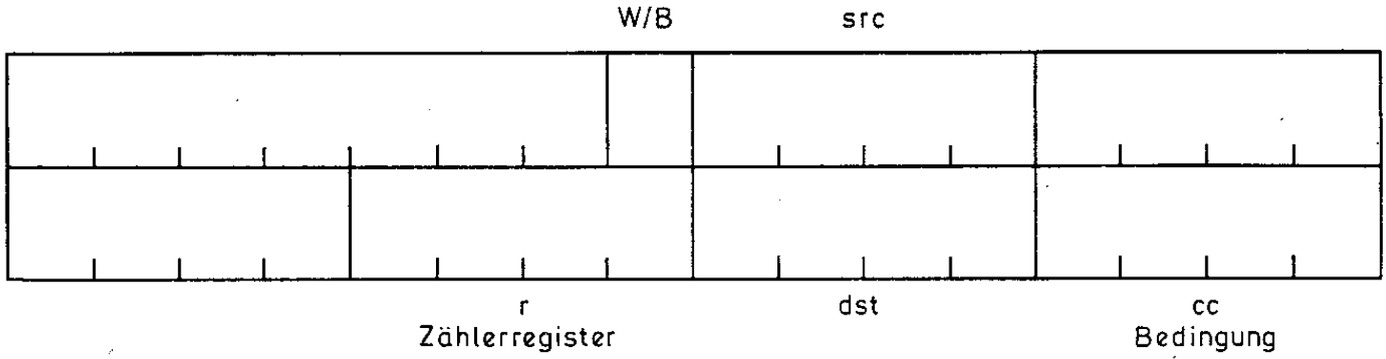
Beispiele für verschiedene Befehlsformate:

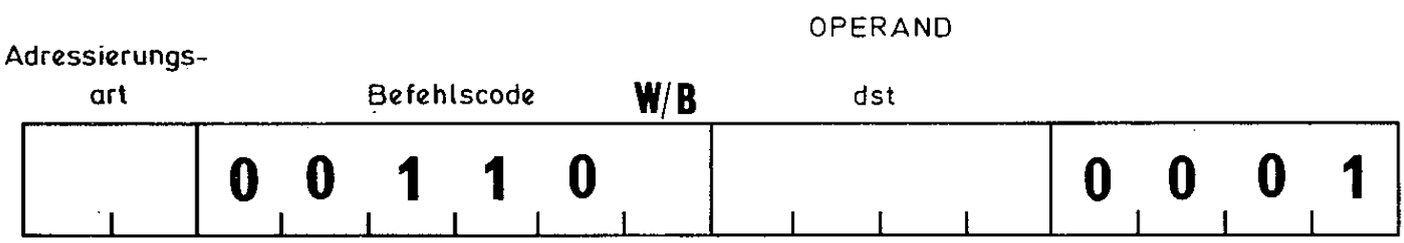


Laden (Doppelwort)
Ziel: Register, Quelle: Basis-Adressierung

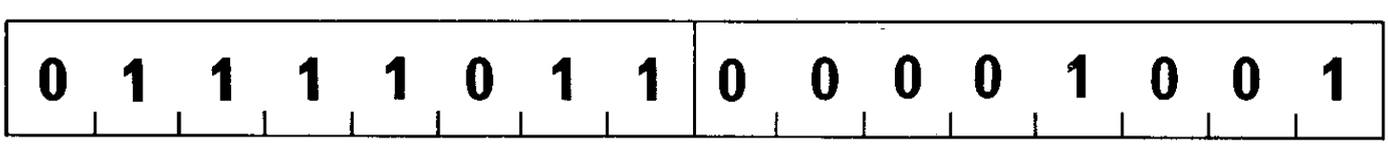


CPSDR, CPSIR (Stringvergleich)

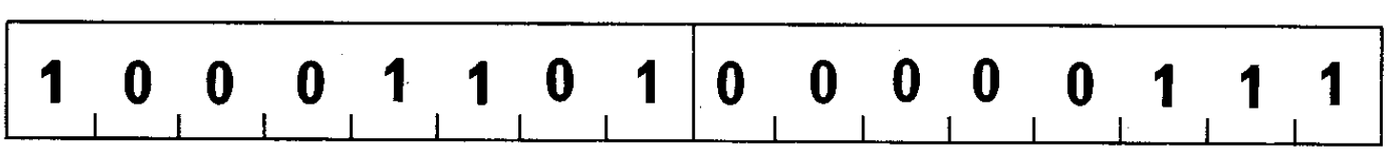




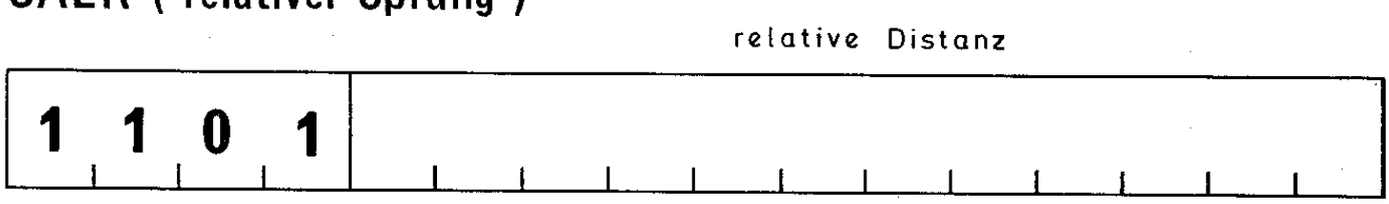
MRES (Multi-Mikro-Reset)



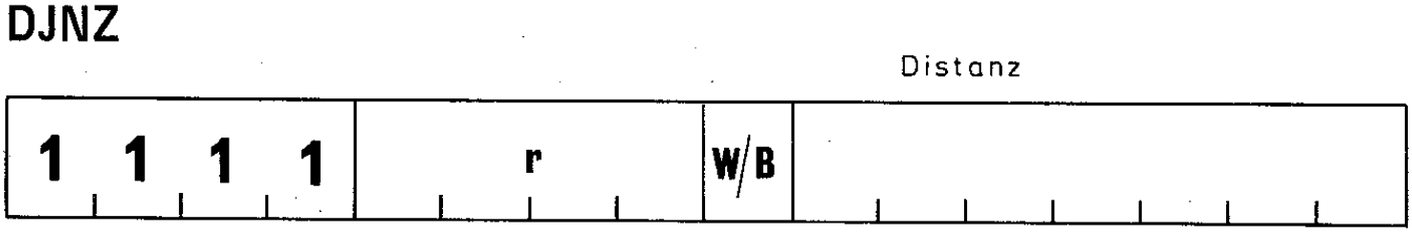
NOP (keine Operation)



CALR (relativer Sprung)



DJNZ



3.3 ASSEMBLER, BEFEHLSELEMENTE, SYMBOLE UND OPERATIONSZEICHEN

Die Z 8000-Befehle in der Assemblersprache anzuwenden, wird eine vielfach angewandte Methode sein. Für das jeweilige Z 8000-System ist hierfür ein Maschinensprache-Compiler notwendig, der allgemein als **Assembler** bezeichnet wird. Wenn dies, zugegeben, auch ein Privileg größerer Z 8000-Maschinen ist, wird an dieser Stelle auf die Behandlung und den Umgang mit dem Z 8000-Assembler-Syntax eingegangen. Die Programmierung des Z 8000 ist selbstverständlich in sedezimaler Form (HEX) durchführbar, wie für jeden anderen Prozessor; diese primitivste Form einer Programmierung kommt jedoch nur für Lehrzwecke oder Billig-Systeme in Frage. Die Programmierung des Z 8000 in einer höheren Programmiersprache steht hier nicht zur Diskussion, da in diesem Fall der Prozessor nicht mehr im Vordergrund steht.

Einer der ersten großen Programme, die für den Z 8000 entwickelt wurden, ist ein Übersetzerprogramm für die Z 8000-Mnemonics. Mnemonics sind die Befehlkurzbezeichnungen des Prozessors. Ein Assembler besteht aus mehreren Elementen: Mnemonic, Quell- und Zielangabe für die verwendeten Operanden, symbolische Kurzzeichen für die Bezeichnung von Werten und letztlich gewisse Labels für symbolische Adressen. Ferner muß der Assembler alle auf die CPU bezogenen Bedingungen enthalten. Diese Bedingungen besagen, welche Register verwendet werden dürfen (Byte-, Wort-, Doppel- und Vierfach-Register), welche Adressierungsarten für den jeweiligen Befehl gültig sind, welcher Modus (segmentiert, nicht segmentiert) vorgegeben und gestattet ist und noch eine Reihe weiterer Punkte. Daraus ist erkennbar, daß ein Assembler ein ziemlich kompliziertes Programm ist. Die Aufgabe des Assemblers ist es, eine Assembler-Programmliste Zeile für Zeile auf syntaktische Fehler zu überprüfen und dann die Übersetzung in Ein- oder Zweiwort-Maschinencodes (bis 5 Worte) auszuführen.

Beispiel: Den Assembler-Syntax

```
ADDL   RR2, #%032F
```

übersetzt das Compilerprogramm in

1. Wort:

```
      15 . . .                    . . . 0
MSB – 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 – LSB
```

2. Wort:

```
MSB – 0 0 0 0  0 0 1 1 0 0 1 0 1 1 1 1 – LSB
```

Das erste Wort enhält das eigentliche Befehlswort, dessen Einteilung aus Bild 3.2 ersichtlich ist; das zweite Wort enthält den Daten-Operanden #%032F in sedezimaler Schreibweise, womit in diesem Befehl zugleich die unmittelbare Adressierungsart angedeutet ist.

Beim Eingeben der Assemblerprogrammliste in den Speicher sollte eine minimale Editierfunktion vorhanden sein. Das bezieht sich auf eine Korrektur während der Eingabe und eine nachträgliche Korrektur irgendeiner Zeile. Dieser Editor gehört nicht zum Assembler. Das Editorprogramm ist Teil des Betriebssystems.

Ebenso kann eine gewisse Bedeutung ein sogenannter "Linker" erlangen.

Wird ein Assemblerprogramm nur mit einer Zeilennummer versehen, ist eine Einordnung des Programms in den bestehenden Speicherraum notwendig. Der Linker hat die Aufgabe, das Programm in den vorgesehenen Speicherbereich zu verschieben, in dem es zur Anwendung kommen soll. Wichtig ist ein Linker dann, wenn E/A-Operationen im Programm eingebaut sind. Die fest definierten E/A-Adressen für die Ports werden vom Linker eingesetzt, so daß der Programmierer nur Labels in seinem Programm zu setzen braucht. Der Linker ordnet den E/A-Operationen gegebenenfalls automatisch nur bestimmte Port-

adressen zu, die für das Programm reserviert sind. Ein Linker ist grundsätzlich nur dann notwendig, wenn entwickelte Programme auf verschiedenen Z 8000-Systemen betrieben werden sollen, die über unterschiedliche Speicherfeldkonzeptionen und E/A-Ports verfügen. Vorrangig ist dies für segmentierte Systeme bedeutend, die mit Programmen aus vielerlei Quellen betrieben werden.

Abschließend soll zu den Unterstützungsprogrammen zum Assembler noch der Typ des Makro-Assemblers genannt sein. Makro-Assembler betreffen bereits exklusive Ausführungen solcher Übersetzerprogramme. Sie erlauben, einmal geschriebene Programmteile (abgeschlossene Teilprogramme in Assembler) mit einem Label zu versehen, so daß sie bei einem weiteren Gebrauch nur mit einem spezifizierten Wort aufzurufen sind. Dieses Wort ist als Makrobefehl zu bewerten und ist beliebig in eine Assemblerroutine einzubauen. Je nach Art kann der Assembler eine Reihe solcher Makros erkennen und in die Routine mit einschleifen. Der Sprung erfolgt relativ oder direkt. In den Makro-Labels ist zugleich ein definierter Rücksprung mit enthalten. Die Definition eines Makros wird im Assembler in einer separaten Tabelle aufbewahrt. Hat ein Makro-Assembler beispielsweise 10 Labels zu vergeben, können 10 Teilprogramme in einem Assemblerprogramm mit Makrobefehlen versehen werden. Die Makrobefehle gelten dann nur für ein bestimmtes Assemblerprogramm. Die gleichen 10 Labels sind für ein anderes Programm wieder neu zu definieren.

Nachdem für den Begriff Assembler etwas weiter ausgeholt wurde, wird zum Kern zurückgesprungen. Das Mnemonic ADDL wird als ASCII-Zeichenfolge in einer Tabelle aufgesucht. Das L am Schluß betrifft eine Langwort-Operation. Ist ADDL für gültig anerkannt worden, wird an Bit 13 . . . 8 des ersten Wortes der Wert 010110 als Opcode gefunden. Nach der Mnemonic erfolgt die Zielangabe. RR2 ist die Bezeichnung eines Doppelregisters im Z 8000-Registersatz. Man muß in das src- bzw. dst-Feld einen Binärwert dem jeweilig betroffenen Register zuordnen. Folgende Tabelle zeigt die Zuordnung der betreffenden Register zu einem Binärwert, der im Befehlsformat als jeweilige Kennung kodiert ist.

Registertyp				
Vierfach-	**Doppel-**	**Wort-**	**Byte-Register**	**Binärcodierung**
RQ 0	RR 0	R 0	RH 0	0 0 0 0
		R 1	RH 1	0 0 0 1
	RR 2	R 2	RH 2	0 0 1 0
		R 3	RH 3	0 0 1 1
RQ 4	RR 4	R 4	RH 4	0 1 0 0
		R 5	RH 5	0 1 0 1
	RR 6	R 6	RH 6	0 1 1 0
		R 7	RH 7	0 1 1 1
RQ 8	RR 8	R 8	RL 0	1 0 0 0
		R 9	RL 1	1 0 0 1
	RR 10	R 10	RL 2	1 0 1 0
		R 11	RL 3	1 0 1 1
RQ 12	RR 12	R 12	RL 4	1 1 0 0
		R 13	RL 5	1 1 0 1
	RR 14	R 14	RL 6	1 1 1 0
		R 15	RL 7	1 1 1 1

Wählt man für einen Befehl die Registeradressierung, ist aus der Tabelle der 4-bit-Wert für das jeweilige Register zu entnehmen und im Befehlsformat einzusetzen. Wählt man für die Quellangabe die unmittelbare oder die direkte Adressierungsart, ist im src-Feld 0000 einzutragen. Wählt man allgemein die indirekte, die indizierte, die Basis oder die Basis indizierte Adressierungsart, darf das betreffende dst- oder src-Feld nicht null sein. Ein Assembler-Syntax des Additions-Befehls in indizierter Adressierung art wird folgendermaßen geschrieben:

ADDL RR2, % FF11 (R9)

Für diesen Fall wird im Quellfeld der Wert 1001 eingetragen. In R9 steht die Distanzadresse.

Ein anderes Beispiel:

SET R2, @ R11 (Bit setzen)

Der zu bearbeitende Operand steht mit seiner Adresse in Register R11. Im Quellfeld wird hierfür der Wert 1011 eingetragen.

Der Z-Befehlssatz enthält überdies noch einen weiteren Befehlstyp, der an dieser Stelle unbedingt betrachtet werden muß. Diese Befehle haben im Befehlsformat ein 4-bit-Bedingungsfeld mit dem Kurzzeichen "cc". **Bild 3.13** zeigt das Bedingungsfeld im Format des Doppelwort-Befehls CPSIR (Vergleich Zeichenkette). Dieses Feld ist nach dem entsprechenden Code der Bedingung zu belegen. Weitere Befehle sind JP, JR, RET und TCC (Sprung absolut, Sprung relativ, Rücksprung, Test Bedingung). Folgender Befehl sei als Beispiel genannt:

Kürzel	**Bedeutung**	**betroffenes Flag**	**Wert für cc**
ohne Bz.	immer nicht erfüllt	— —	0 0 0 0
ohne Bz.	immer erfüllt	— —	1 0 0 0
Z	Null-Flag gesetzt	Z = 1	0 1 1 0
NZ	Null-Flag nicht gesetzt	Z = 0	1 1 1 0
C	Carry-Flag gesetzt	C = 1	0 1 1 1
NC	Carry-Flag nicht gesetzt	C = 0	1 1 1 1
PL	Vorzeichen-Flag nicht gesetzt	S = 0	1 1 0 1
MI	Vorzeichen-Flag gesetzt (—)	S = 1	0 1 0 1
NE	Operand nicht gleich	Z = 0	1 1 1 0
EQ	Operand gleich	Z = 1	0 1 1 0
OV	Überlauf aufgetreten	P/$\overline{V}$ = 1	0 1 0 0
NOV	kein Überlauf aufgetreten	P/$\overline{V}$ = 0	1 1 0 0
PE	Parität gerade	$\overline{P}$/V = 1	0 1 0 0
PO	Parität ungerade	$\overline{P}$/V = 0	1 1 0 0
GE	größer als oder gleich	(S XOR V) = 0	1 0 0 1
LT	kleiner als	(S XOR V) = 1	0 0 0 1
GT	größer als	(Z OR (S XOR V)) = 0	1 0 1 0
LE	kleiner als oder gleich	(Z OR (S XOR V)) = 1	0 0 1 0
UGE	größer als oder gleich (ohne Vorzeichen)	C = 0	1 1 1 1
ULT	kleiner als (ohne Vorzeichen)	C = 1	0 1 1 1
UGT	größer als (ohne Vorzeichen)	((C=0) AND (Z=0)) = 1	1 0 1 1
ULE	kleiner als oder gleich (ohne Vorzeichen)	(C OR Z) = 1	0 0 1 1

JP % FF11 (unbedingter, absoluter Sprung)

wird übersetzt in

1. Wort
MSB – 0101111100001000 = % 5E08
(dst: 0000, cc: 1000)

2. Wort
MSB – 1111111100010001 = % FF11

Die direkte Adressierung erfordert das Nullfeld von dst. Das Bedingungsfeld "cc" hat hier den Wert 1000 (binär) und besagt, daß in jedem Fall gesprungen wird, komme was da wolle. Es gibt eine Reihe von Bedingungen, die man einsetzen kann.

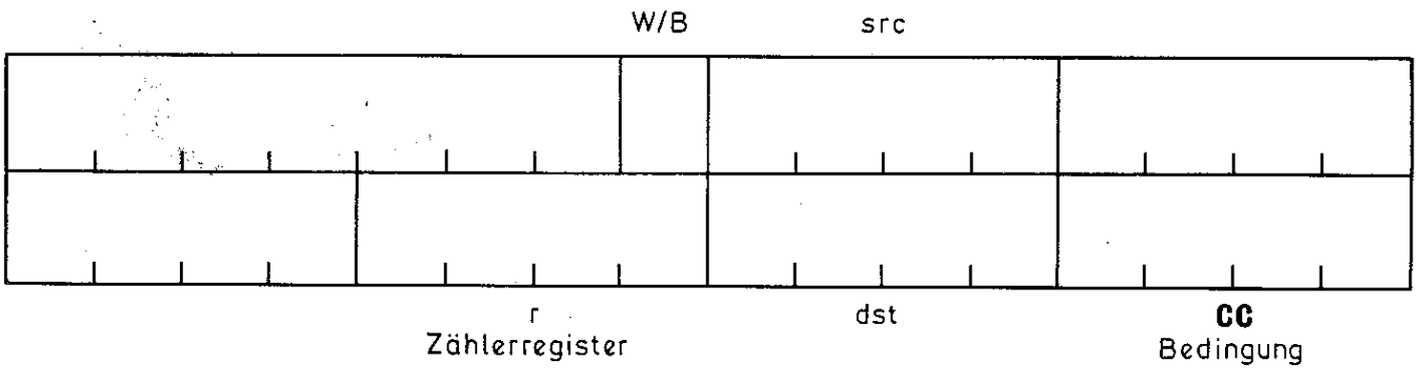


Bild 3.13

Beispiel:

JP NZ, % FF11 (Sprung, wenn Operand nicht Null)

JP MI, % FF11 (Sprung, wenn Operand negativ)

Wie hier zu bemerken ist, wird vor der Sprungadresse ein Kürzel gesetzt, das die Bedingung andeutet. Im ersten Beispiel des unbedingt absoluten Sprungs fehlt das Kürzel. Das fehlende Kürzel beschreibt die Bedingung als absolut.

Die folgende Tabelle gibt die verschiedenen Bedingungscodes an, die in den jeweiligen Befehlen gesetzt werden können. Als Zeichen für eine Erfüllung einer Bedingung gelten die einzelnen Flags im FCW-Register der CPU.

Die cc-Tabelle beinhaltet 22 codierte Bedingungen. Die Permutation von 4 Stellen lassen nur 16 zu. Es ergeben sich deshalb gleichbedeutende Codes für einige Bedingungen. Folgende Kürzel haben den gleichen Wert für cc:

Z ↔ EQ,	NC ↔ UGE
NZ ↔ NE,	OV ↔ PE
C ↔ ULT,	NOV ↔ PO

Alle Befehle mit einem cc-Feld in ihrem Format behandelt der Assembler mit beiden Kürzelangaben. Das betreffende Kürzel gleichen Codes, aber unterschiedlicher Schreibweise wird bezüglich einer vorausgegangenen Operation arithmetischer oder logischer Art eingesetzt. Es ist ausschließlich vom Programm abhängig.

Bis zu dieser Stelle dieses Kapitels kann man fast schon eine komplette Zeile eines Assembler-Befehls für die CPU entwerfen. Es fehlen zu einer syntaktisch richtigen Schreibweise noch einige kleine Operationszeichen, deren Bedeutung man kennen sollte. Die Assembler-Syntaxregeln schreiben vor, daß Daten und Adressen, dezimale und sedezimale Wer-

te entsprechend zu kennzeichnen sind. In dem Beispiel

```
ADDL    RR2, #%032F
```

sind zwei Symbole enthalten, die die Kennzeichnung betreffen. Eines davon ist ein reines Operationszeichen; das andere ist ein Symbol für ein Zahlensystem. Irgendwelche Daten werden mit einem voranstehenden Gatter versehen:

```
#Daten
```

Die Daten betreffen vorrangig einen Zahlenwert für einen Operanden. Sie betreffen auch einen Indexwert (Anfangswert für einen Zähler, der z.B. zyklisch dekrementiert wird). Die Daten betreffen nach dem Zeichen "#" **keinesfalls** einen Wert, der in irgendeiner Form mit einer Adresse zu tun hat. Der Datenwert als Zahl betrachtet, bezieht sich auf ein Zahlensystem.

```
#458
```

gibt für den Datenwert eine Dezimalzahl an, die der Assembler in der Form akzeptiert. Die CPU arbeitet jedoch überwiegend mit binär verschlüsselten Daten. Somit ist die Darstellung von Daten in dem verbreiteten sedezimalen Zahlensystem üblich und wird vom Assembler in erster Linie akzeptiert.

```
#%032F
```

bezeichnet einen Wert im sedezimalen Zahlensystem. # % 032F entspricht #815 im dezimalen System. Nach den Assembler-Syntaxregeln entspricht das Prozent-Zeichen "%" vor dem Zahlenwert, daß es sich um einen sedezimalen (oder hexadezimalen) Wert handelt. Für % 032F wird auch 032FH geschrieben, jedoch **nicht in Assembler-Programmlisten.** Es gilt auch eine binäre Ausgabe:

```
#%032F = #%(2) 1100101111
```

Ein weiteres Zeichen, das bereits mehrfach auftrat, ist das kleine a mit Schlinge "@", das von Programmierern auch symbolisch als "Klammeraffe" bezeichnet wird. Dieses Zeichen gibt die Adresse für einen Operanden an, der von einem Register im Speicher adressiert wird. Tritt dieses Zeichen in einer Befehlszeile auf, deutet es auf die indirekte Adressierungsart hin. Für den Z 8000-Assembler gilt z.B.

```
MULTL   RQ0, @R8
```

als Schreibweise im nicht segmentierten Modus und

```
MULTL   RQ0, @RR8
```

als Schreibweise für den segmentierten Modus. In R8 steht die Adresse des noch im Speicher befindlichen zweiten Operanden. In RR8 steht die Adresse eines Operanden in einem segmentierten Speicherfeld, d.h. in R8 steht die Segmentnummer und in R9 die Offsetadresse. Folgende Befehle zeigen weitere Beispiele einer gültigen Schreibweise für die indirekte Adressierungsart:

```
LDM     R0,     @RR14
```

```
LDM   @RR14, R0 #16
PUSH  @RR2,  #%032F
NEG   @RR2
LDIR  @RR2,  @RR4, R0
```

Ein weiteres Zeichen der Form "$" ist in die Z 8000-Assemblersprache einbezogen, findet jedoch nicht so häufige Anwendung wie das vorhergehende. Dieses Zeichen bezieht sich auf die relative Adressiegungsart. "$" symbolisiert den momentanen Programmzählerstand (generell nur für den Offset). Einige Befehle mit ausschließlich relativer Adressierung erfordern die Anwendung dieses Zeichens. Das sind CALR, LDAR, DJNZ, LDR und JR. Folgende Schreibweise ist gültig:

```
LDR  R10,  $+20
JR   NZ,   $-20
JR   OV,   $+K
```

Einschränkend sei hier bemerkt, daß beim Arbeiten mit Labels die Adreßdifferenzangabe und damit das Zeichen nicht benötigt wird.

Zuletzt wird eine bereits angedeutete Schreibweise von Kurz- und Langsegment-Adressen differenzierend dargestellt. In Bild 3.7 wurde auf die registermäßige Unterscheidung unterschiedlicher Segmentlängen hingewiesen. Ist das SEG-Steuerbit im FCW-Register auf 1 gesetzt (Z 8001), müssen Operanden oder UPRO-Aufrufe eine Segmentadresse aufweisen. Dies wird im allgemeinen folgendermaßen geschrieben:

≪Segment-Nr.≫ Offset

In einem Befehl angewandt, ist folgende Schreibweise gültig:

```
LD   R4, ≪75≫ %00B1
```

Dies bezeichnet ein Segment mit der Größe 256 Byte < Segment-Offset ≤ 64 KB. Soll die Quelle in einem Kurzsegment bezeichnet werden 1 ≤ Offset ≤ 256, wird die Adreßangabe mit senkrechten Strichen versehen.

```
LD   R4, |≪74≫ %00B1|
```

Für Kurzsegmente ist auch die Schreibweise

```
LD   R4, |≪74≫ %B1|
JP   |≪20≫ %11|
```

erlaubt. Nicht erlaubt ist

```
LD   R4, |≪74≫ %100|
```

weil die Offsetadresse % 100 das 8-bit-Format überschreitet. Kurzsegmentangaben werden vom Assembler nur in der direkten und indizierten Adressierungsart akzeptiert.

Beispiel:

```
LD   R4, |≪74≫ %B1| (R10)
```

Die Klammer um das Register gibt die Distanz bzw. den Indexwert bei der indizierten, Basis- oder Basis indizierten Adressierung an. Für einen Segmentzugriff in ein Langoffset-Segment wird in der indizierten Adressierungsart gültig geschrieben:

```
LD   R4, ≪75≫ %01B1 (R10)
```

Es sollte hier nochmals darauf hingewiesen werden, daß die explizite Segment-

adreßangabe als selten zu bezeichnen ist, weil man die Sprungdistanz oder die absolute Sprungadresse ausrechnen muß. Normalerweise sollte man diese Aufgabe einem Linker überlassen. Besteht dieses Softwarehilfsmittel im System, werden für Operanden oder Programmodule eine Kennung gesetzt, für die der Linker die gültige Adresse ermittelt und einsetzt.

```
LD   R4, | KONSTANTE |      direkt
LD   R5, | FAKTOR | (R10)  indiziert
```

oder bei Langoffset-Segmenten

```
LD ≪DATEI≫ % 01B1, R6   Ziel direkt
```

Mit der Darstellung vorangegangener verschiedener "Kleinigkeiten" ist damit die minimale Hilfestellung gegeben, um Assemblerprogramme syntaktisch richtig schreiben zu können. Daß für ein erfolgreiches Entwerfen von Z 8000-Assemblerprogrammen mehr notwendig ist, als nur die Regeln einzuhalten, versteht sich von selbst.

3.4 Z 8000-BEFEHLSGRUPPEN

Die Z 8000-Befehle lassen sich in neun Gruppen einteilen, die jeweilig eine Anwendungsrichtung sehr genau spezifizieren. In Abschnitt 3.2 wurde bereits ein Überblick gegeben, wie die Gruppen überschrieben sind. Ferner wird darunter angedeutet, welche Befehlsart in die jeweilige Gruppe eingereiht ist. Aus diesem Grund kann es auch für einen weniger betrauten Z 8000-Anwender keine Schwierigkeiten bedeuten, sich in dem Befehlsspektrum zurechtzufinden. Aus der Gliederung zu Beginn des Buches ist die Strukturierung der Befehle ebenfalls ersichtlich.

3.4.1 Laden und Ändern von Register- und Speicherinhalte

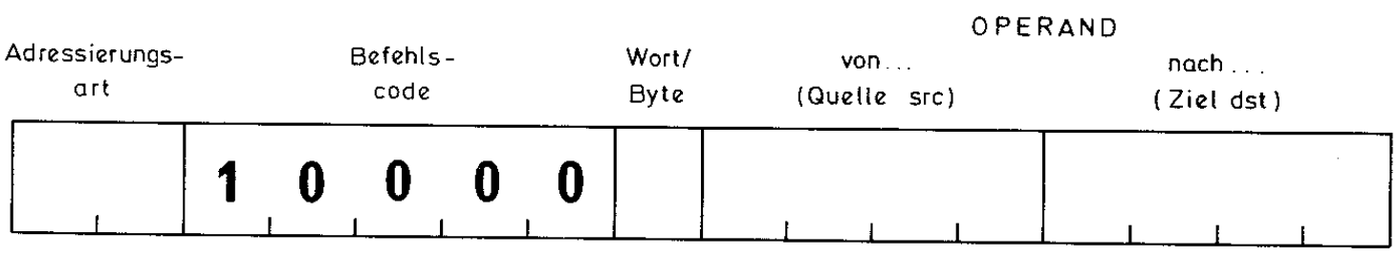
Die Befehle betreffen das Laden eines Registers mit Daten von einem anderen Register oder aus dem Speicherwerk; ferner das Laden einer Speicherstelle aus dem Register und zuletzt das Laden von Registern oder dem Speicher mit unmittelbar im Befehl stehenden Daten. Folgende Tabelle gibt die Befehle an, die dieser Gruppe angehören:

Befehl	Operanden		Kurzbezeichnung
	Ziel	Quelle	
CLR CLRB	•		Rücksetzen von Register oder Speicherstelle
EX EXB	•	•	Austausch des Inhalts; Register-Register, Register-Speicherstelle
LD LDB LDL	•	•	Ladeoperation; Register-Register, Register-Speicherstelle, Speicherstelle-Register
LDA	•	•	Adresse in Register laden
LDAR	•	•	Adresse in Register laden (relativ)

LDK	●	●	laden Konstante (0 ⩽ Wert ⩽ 15)
LDM	● (Anzahl)	●	Operandentransfer (bis 15 Operanden)
LDR LDRB LDRL	●	●	laden sowohl Register als auch Speicherstelle mit relativer Adresse Berechnung: PC ± Distanz
POP POPL	●	●	laden in Register vom Stapel
PUSH PUSHL	●	●	laden in Stapel von Register

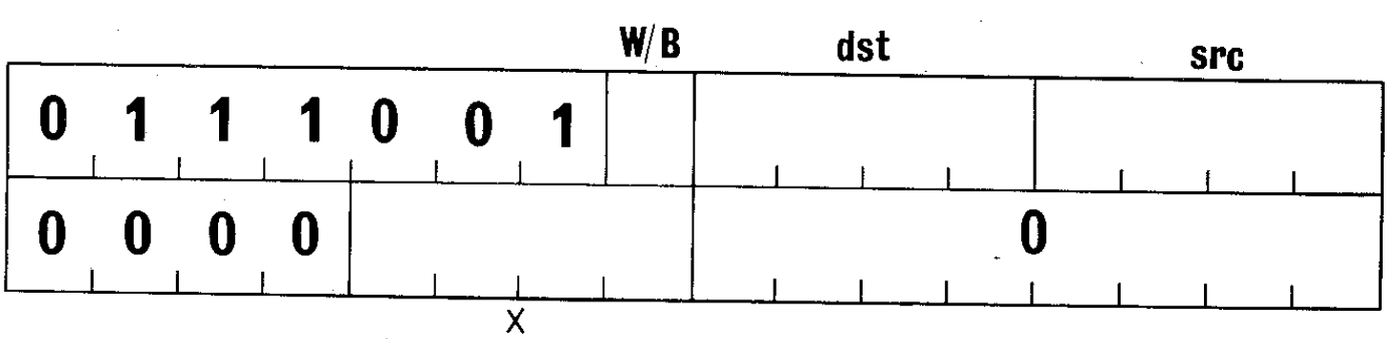
Beispiele einzelner Assembler-Befehle:

LD



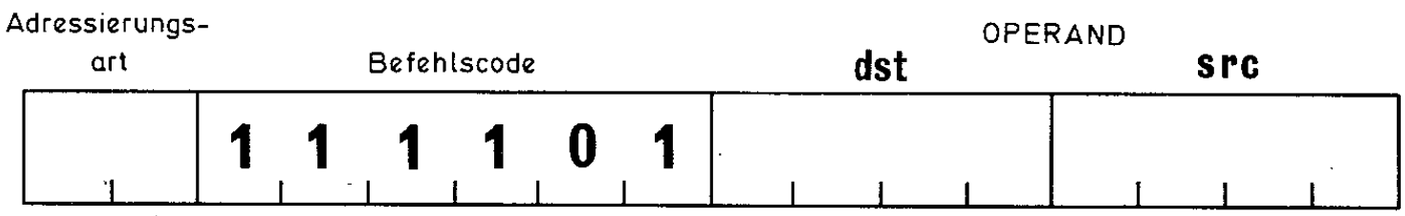
generell mögliche Adressierungsarten:
dst: R, IR, DA, X, BA, BX
src: R, IM, IR, DA, X, BA, BX

Adressierungsart	dst	src
10	R	R
00	R	IM (src = 0)
00	R	IR (src ≠ 0)
01	R	DA (src = 0)
01	R	X (src ≠ 0)



Adressierungsart	dst	src
	BX	R

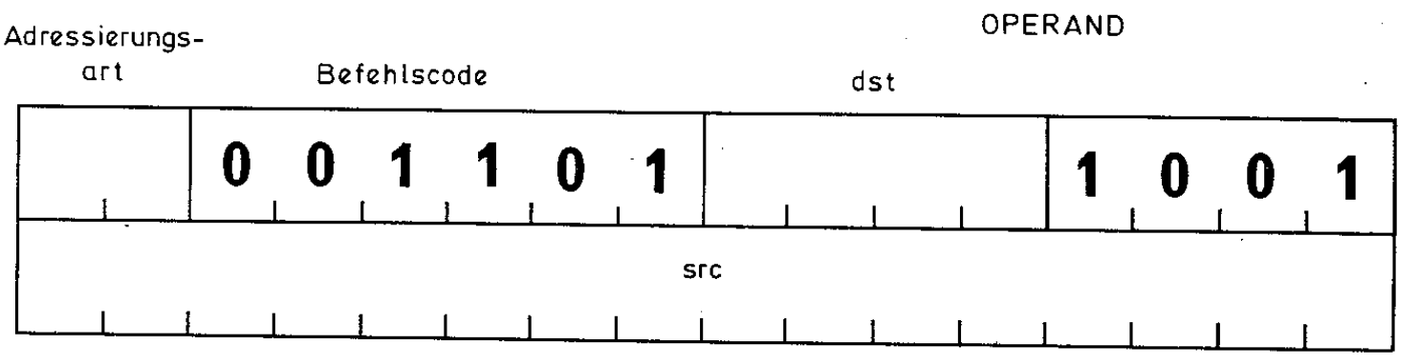
LDK



im src Wert für Konstante angeben (src → 0 . . . 15, binär)

Adressierungsart	dst	src
10	R	IM

PUSH



Stapelzeiger wird automatisch dekrementiert (—2 bei Wort; —4 bei Langwort)

Adressierungsart	dst	src
00	IR	IM

3.4.2 Arithmetische Operationen

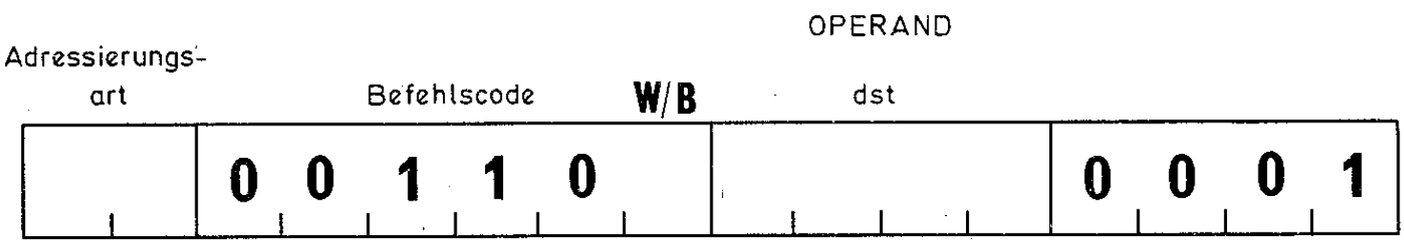
Den Kern der arithmetischen Operationen bilden die vier Grundrechnungsarten, die die CPU generell mit 16-bit- und 32-bit-Operanden ausführt. Addition und Subtraktion werden auch im Byteformat ausgeführt. Sie werden über eine weitere Variation auch mit einem vorher gesetzen Übertragsbit ausgeführt, so daß z.B. 64-bit- oder 128-bit-Additionen bzw. Subtraktionen leicht berechnet werden können. Die Vergleichsoperationen haben Byte-, Wort- und Doppelwortformat. Sie beruhen üblicherweise auf einer Subtraktion. Dekrementieren und Inkrementieren (Zweierkomplement-Addition in absoluten Schritten von 1 . . . 16) werden für Byte- und Wortoperanden ausgeführt. Die Negation ergibt, ebenso für Byte- und Wortoperanden, das echte Zweierkomplement (110 → 1010). Aus den Ergebnissen von Additionen und Subtraktionen binärer Art lassen sich reine BCD-Werte mit dem DAB errechnen. Folgende Tabelle gibt die Befehle für arithmetische Operationen an:

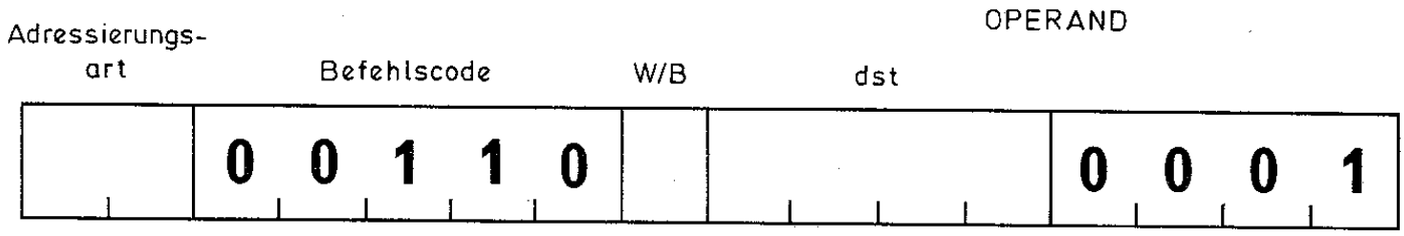
Befehl	Operanden Ziel	Operanden Quelle	Kurzbezeichnung
ADC ADCB	•	•	binäre Addition mit Einbeziehen eines gesetzten C-Flags
ADD ADDB ADDL	•	•	binäre Addition (C-, Z-, S- und V-Flag betroffen, D- und H-Flag eventuell bei ADDB)
CP CPB CPL	•	•	Vergleich (C-, Z-, S- und V-Flag betroffen)
DAB	•		Binär in BCD-Umrechnung (C-, Z-, C-)
DEC DECB	•	•	Dekrementieren eines absoluten Wertes, variabel von 1 . . . 16
DIV DIVL	•	•	binäre Division (C-, Z-, S- und V-Flag betroffen)
EXTS EXTSB EXTSL	•		Vorzeichenübertragung in höherwertige Bytes bei mehrfachen Arithmetik-Operationen
INC INCB	•	•	Inkrementieren eines absoluten Wertes, variabel von 1 . . . 16
MULT MULTL	•	•	binäre Multiplikation (C-, Z-, S- und V-Flag betroffen)
NEG NEGB	•		Negation, echtes Zweierkomplement (C-, Z-, S- und V-Flag betroffen)
SBC SBCB	•	•	binäre Subtraktion mit Einbeziehen eines gesetzten C-Flags
SUB SUBB SUBL	•	•	binäre Subtraktion (C-, Z-, S- und V-Flag betroffen, D- und H-Flag eventuell bei SUBB)

Beispiele einzelner Assembler-Befehle:

CP

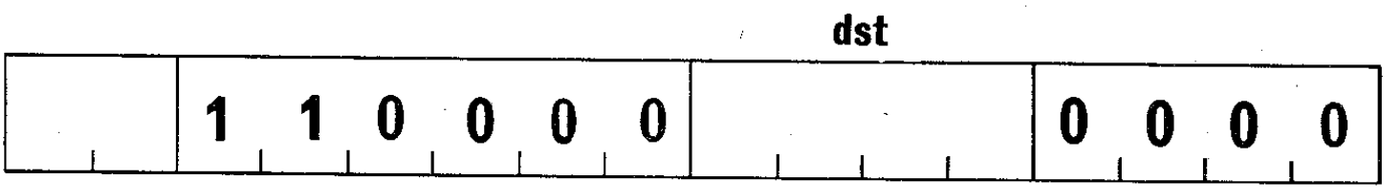
Adressierungsart	dst	src
10	R	R
00	R	IM (src = 0)
00	R	IR (src ≠ 0)
01	R	DA (src = 0)
01	R	X (src ≠ 0)





Adressierungsart	dst	src
00	IR	IM
01	DA	IM (dst = 0)
01	X	IM (dst ≠ 0)

DAB

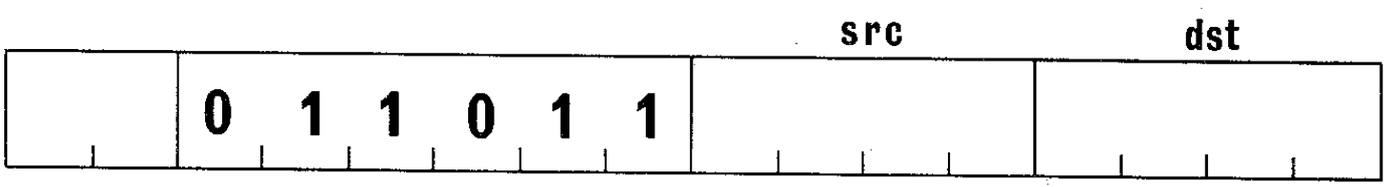


dst ← DA dst

Adressierungsart	dst
10	R

Beachte:
Die Operation ist undefiniert, wenn der umzuwandelnde Operand nicht aus einer vorhergehenden Addition bzw. Subtraktion von BCD-Digits stammt.

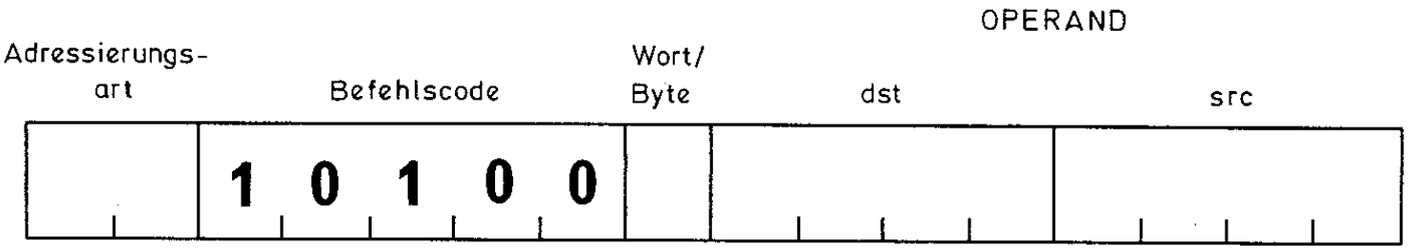
DIV



Bei diesem Befehl werden die Flags Z, S, V beeinflußt.

Adressierungsart	dst	src
10	R	R
00	R	IM (src = 0)
00	R	IR (src ≠ 0)
01	R	DA (src = 0)
01	R	X (src ≠ 0)

INC



Bei diesem Befehl werden die Flags Z, S, V beeinflußt.

Adressierungsart	dst	src
10	R	IM
00	IR	IM
01	DA	IM (dst = 0)
01	X	IM (dst ≠ 0)

3.4.3 Logische Operationen

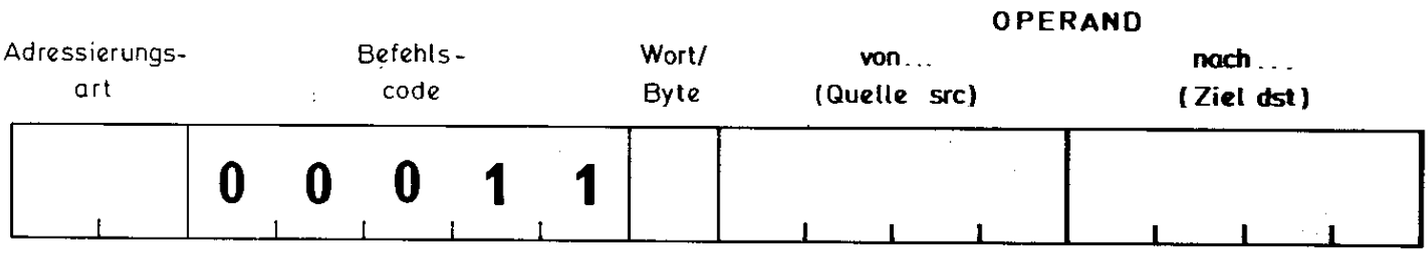
Der Z 8000 ist ein logischer Prozessor in seiner strukturiert, hardwaretechnischen Logik, wie auch in seinen Befehlen. Er führt die wichtigsten Booleschen Operationen programmierbar aus. Damit ist die UND-, ODER- und die exklusive ODER-Funktion angesprochen. Ferner unterscheidet er sehr wohl zwischen der echten und der unechten Komplementbildung. Im dualen Zahlensystem wird die Bildung des unechten Komplements darin gebildet, indem von einem Binärwert die Nullen durch Einsen ersetzt werden und umgekehrt (0110001 → 1001110). Diese Funktion erfüllt der Befehl COM. Die echte Komplementbildung (Zweierkomplement) wird hingegen von dem Befehl NEG ausgeführt, der im vorangehenden Abschnitt genannt wurde. Der Befehl NEG unterscheidet sich nur darin, indem zum Ergebnis immer eine 1 hinzugezählt wird (0110001 → NEG → 1001110 + 1 = 1001111). Aus diesem Grund ist NEG kein logischer, sondern ein arithmetischer Befehl. Als Ausnahme gilt der Wert 8000H bzw. 80H, der von COM und NEG gleich ausgeführt wird. Das ist jedoch rein logisch bedingt. Der Testbefehl mit gleichnamiger Bezeichnung TEST dient für eine ODER-Verknüpfung eines Operanden mit einem gleich langen Wert, der aus Nullen besteht. Folgende Tabelle gibt die Befehle für die logischen Operationen an:

Befehl	Operanden Ziel	Operanden Quelle	Kurzbezeichnung
AND ANDB	•	•	logische UND-Funktion (Z-, S- und P-Flag (bei ANDB) betroffen)
COM COMB	•		unechtes Komplement (Z-, S- und P-Flag (bei COMB) betroffen)
OR ORB	•	•	logische ODER-Funktion (Z-, S- und P-Flag (bei ORB) betroffen)

TEST TESTB TESTL	•		logische ODER-Verknüpfung mit einem Null-Operand (Z-, S- und P-Flag (bei TESTB) betroffen)
XOR XORB	•	•	logische Exklusiv-ODER-Funktion (Z-, S- und P-Flag (XORB) betroffen)

Beispiele einzelner Assembler-Befehle:

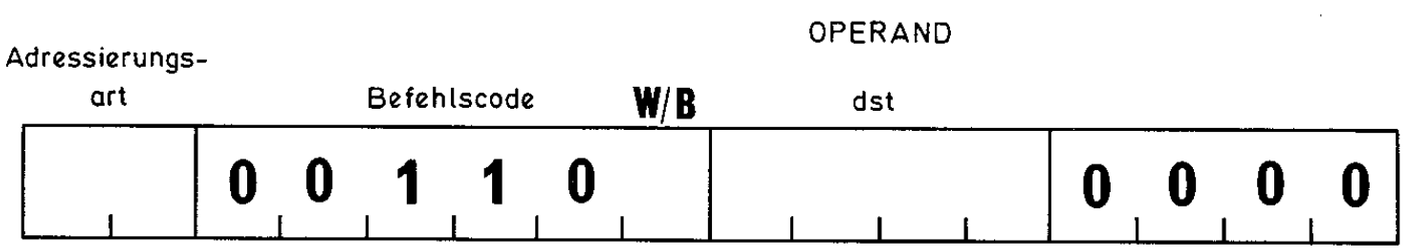
AND



dst ← dst AND src

Adressierungsart	dst	src
10	R	R
00	R	IM (src = 0)
00	R	IR (src ≠ 0)
01	R	DA (src = 0)
01	R	X (src ≠ 0)

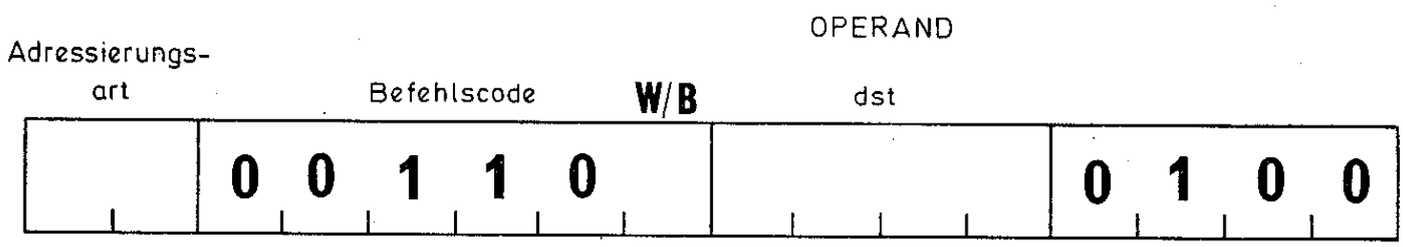
COM



dst ← NOT dst

Adressierungsart	dst
10	R
00	IR
01	DA (dst = 0)
01	X (dst ≠ 0)

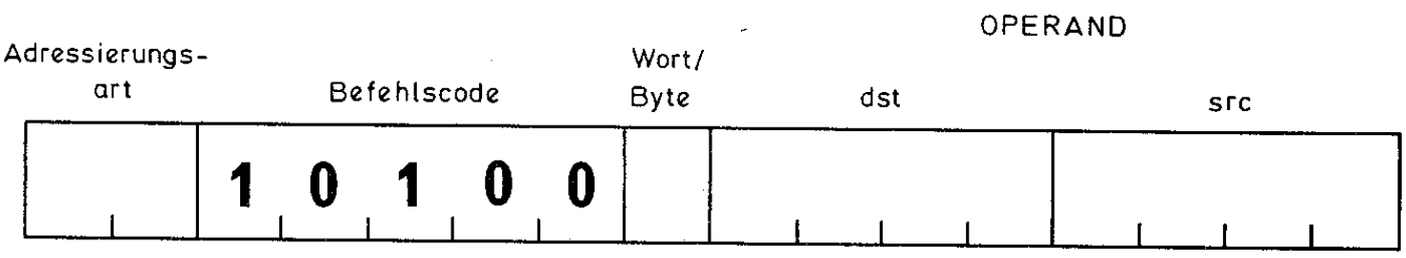
TEST



dst ← OR ϕ

Adressierungsart	dst
10	R
00	IR
01	DA (dst = 0)
01	X (dst ≠ 0)

XOR



dst ← XOR src

Adressierungsart	dst	src
10	R	R
00	R	IM (src = 0)
00	R	IR (src ≠ 0)
01	R	DA (src = 0)
01	R	X (src ≠ 0)

3.4.4 Programmsteueroperationen

Diese Gruppe besteht aus 8 Befehlstypen, die in jedem Fall Sprünge in irgendeinem anderen Programmteil zur Folge haben, sei es bedingt oder unbedingt. Der Programmzähler (PC) ist das davon immer betroffene Register der CPU. Sei es ein genereller Aufruf einer Prozedur (im Betriebssystem oder im Anwenderprogramm) mit einer absoluten Adresse oder einem zum PC relativen Sprungwert oder nur ein Sprung innerhalb eines Anwenderprogramms (absolut oder relativ), die Variation ist vielfältig. "CALLs" sind unbedingte, absolute Sprünge; "CALRs" betreffen unbedingte relative Sprünge. Mit den Befehlen JP und JR erfolgen bedingte Sprünge nach den in Abschnitt 3.3 definierten Gründen. Diese Befehle betreffen Verzweigungen in einem Flußdiagramm und gelten damit als einfache Branch-Befehle. Der DJNZ-Befehl ist ein raffinierter Branch-Befehl, der als Schleifensteuerung verwendet werden kann. Ein beliebiges Register wird als Indexzähler benützt und nach jedem

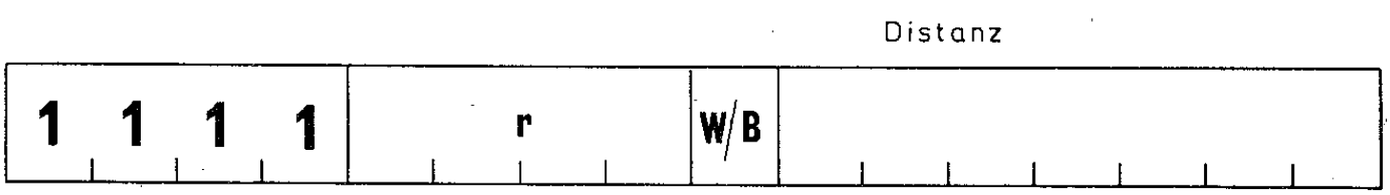
Dekrementieren auf den Wert Null abgefragt. Ist der Index nicht null, wird gesprungen, ansonsten mit dem momentanen PC-Stand fortgesetzt. Die Gruppe schließt zwei Rücksprungbefehle mit ein, von denen RET nach Bedingung erfolgt und IRET eine privilegierte Interrupt-Rückkehr darstellt. Der Befehl "System Call" betrifft einen bedingungslosen Sprung in den System-Modus und damit in einen direkten Betriebssystem-Zugriff. Der Übergang erfolgt in der Art eines Trap-Anerkennungszyklus, d.h. der im Normal-Modus bestandene CPU-Status wird in den Systemstapel transferiert. Folgende Tabelle gibt die Befehle dieser Gruppe an:

Befehl	Operanden Ziel	Operanden Quelle	Kurzbezeichnung
CALL	●		Aufruf UPRO
CALR	●		Aufruf relativ UPRO
DJNZ DBJNZ	● (Register)		Dekrementieren eines Indexwertes und Springen, wenn nicht null
● IRET			Interrupt-Rückkehr
JP	● (Bedingung)		einfacher Sprung
JR	● (Bedingung)		einfacher relativer Sprung
RET	(Bedingung)		Rückkehr aus einem UPRO
SC		●	Aufruf bzw. Anforderung auf eine System-Modus-Routine (Trap-Anerkennungszyklus)

● privilegierter CPU-Befehl

Beispiele einzelner Assembler-Befehle:

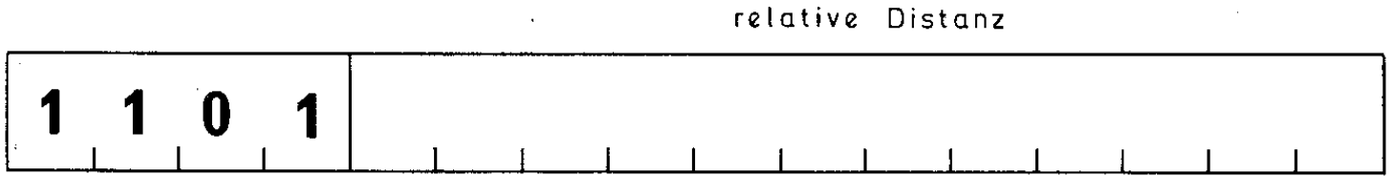
DJNZ



$r \leftarrow r - 1$
$r \neq 0 \rightarrow PC \leftarrow PC - 2 \cdot Distanz$

Register	dst
R	RA

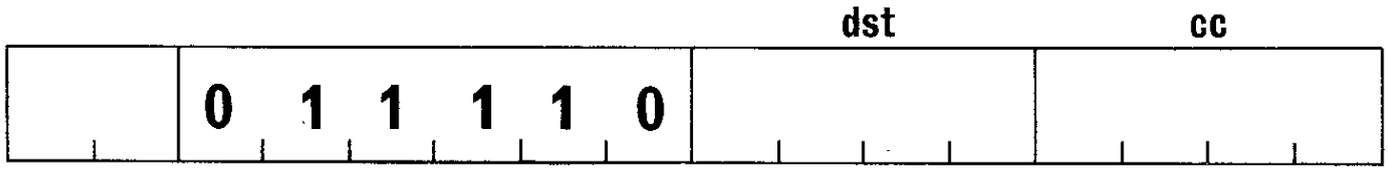
CALR



Stapelzeiger – 2 (– 4 bei SEG = 1)
PC ←PC – 2 · Distanz

dst
RA

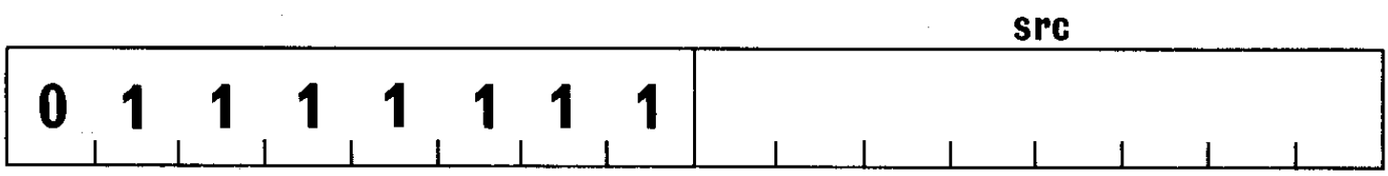
JP



wenn cc erfüllt:
PC ←dst

Adressierungsart	**dst**
00	IR
01	DA (dst = 0)
01	X (dst ≠ 0)

SC



Stapelzeiger (SP) ← SP – 4 (– 6 bei SEG = 1)
@ SP ← Programmstatus (PS)
SP ← $ – 2
@ SP ← Befehl
PS ← System Call PS

Der 8-bit-Wert im src-Feld dient als "Kennung" des Traps, der vom Betriebssystem in einer Tabelle als Bedarf einer bestimmten Serviceroutine erkannt wird.

3.4.5 Operationen für Einzelbit-Manipulation

Diese aus 5 Befehlstypen bestehende Gruppe betrifft das Testen, Setzen, Rücksetzen und Setzen (nach Test) eines Bit. Für die Manipulation eines Bit gilt die statische oder dynamische Art. Man unterscheidet darin die Quelle, in der die Stelle des zu variierenden Bit in einem Byte- oder Wortoperanden definiert ist. Als statische Manipulation gilt die Festlegung der Stelle direkt im Befehl als unmittelbarer Wert.

```
RESB   RL0, #5
```

wird immer die Stelle Nummer 5 (2 hoch 4) im Register RL0 auf Nullsetzen. Wird dagegen statt #5 z.B.

```
RESB   RL0, R11
```

geschrieben, wird der binäre Wert an den untersten 4 bit in Register 11 festlegen, welches Bit in RL0 auf Null gesetzt wird. Dies gilt für alle vier Befehle. Für TSET liegt der Fall noch ein wenig anders. Hier dient das MSB (höchstwertiges Bit) eines Operanden als Testziel. Ist das MSB auf 1 gesetzt, wird nach dem Befehl das S-Flag auf 1 gesetzt sein (bei MSB gleich Null → S gleich Null). Dieser Befehl gestattet in gewissem Rahmen eine Synchronisation von Software. TSET ist in diesem Sinn ein statischer Test. Es sei nebenbei erwähnt, daß während der Ausführung von TSET ein externer $\overline{\text{BUSRQ}}$ (Busanforderung) nicht akzeptiert wird. Allein diese vier Befehle sind für "multiprogramming"- oder "multiprozessing"-Anwendungen sehr brauchbar. Sie können beispielsweise mit den Multiprozessor-Befehlen für verschiedenartige Synchronisations-Mechanismen effektiv eingesetzt werden.

Eine weitere Stütze bringt der TCC-Befehl. Eine vorhergehende Operation hat aus einem bestimmten Grund eines der 7 Flags gesetzt. Mit TCC wird über die jeweilige Bedingung dieses Flag erfaßbar und prüfbar. In einem beliebigen Register wird das LSB (niederwertigstes Bit) gesetzt, wenn die Bedingung erfüllt wurde.

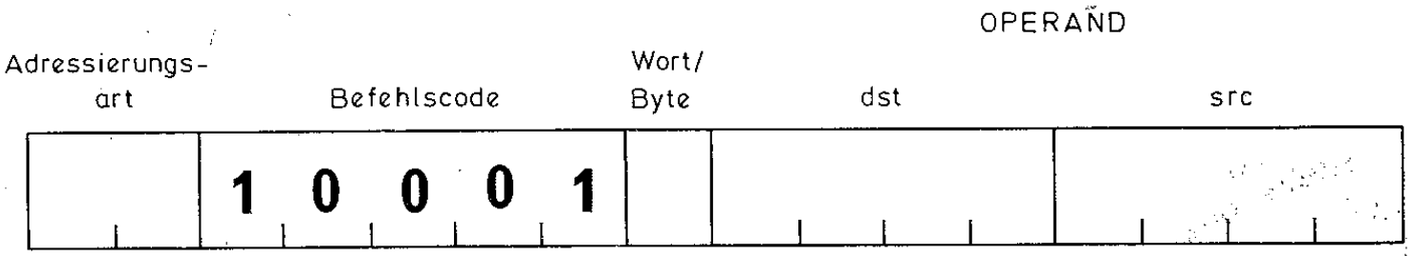
Beispiel:

```
TCCB   NZ, RL0
```

War in RL0 das LSB gleich Null, ist nach der Abfrage des Z-Flags über die Bedingung NZ (siehe Abschnitt 3.3) das LSB von RL0 gleich 1, wenn NZ erfüllt war. Folgende Tabelle faßt die Befehle dieser Gruppe zusammen:

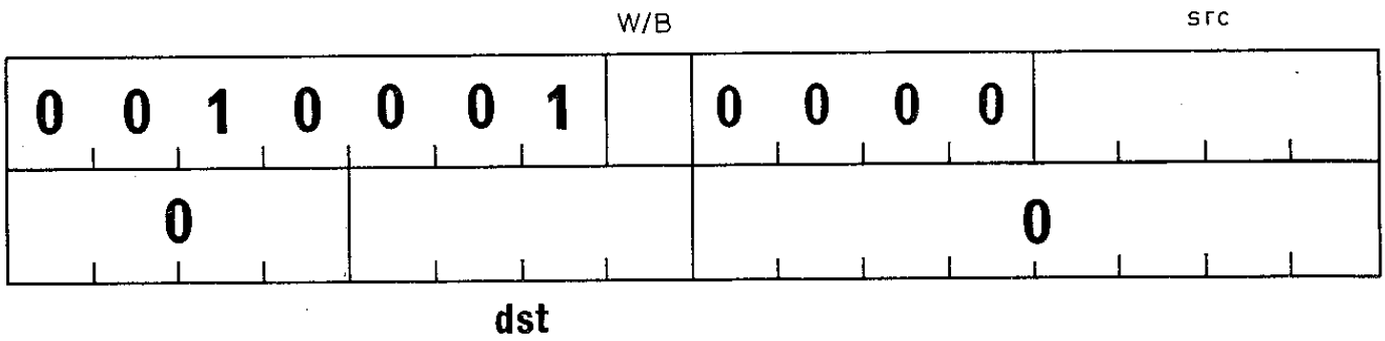
Befehl	**Operanden**		**Kurzbezeichnung**
	Ziel	**Quelle**	
BIT BITB	•	•	Bit-Abfrage statisch, dynamisch (Z-Flag betroffen)
RES RESB	•	•	Bit rücksetzen (statisch, dynamisch)
SET SETB	•	•	Bit setzen (statisch, dynamisch)
TSET TSETB	•		MSB abfragen und S-Flag setzen, wenn MSB gleich 1
TCC TCCB	• (Bedingung)		Flag-Abfrage nach Bedingung

RES



dst (src) ←0

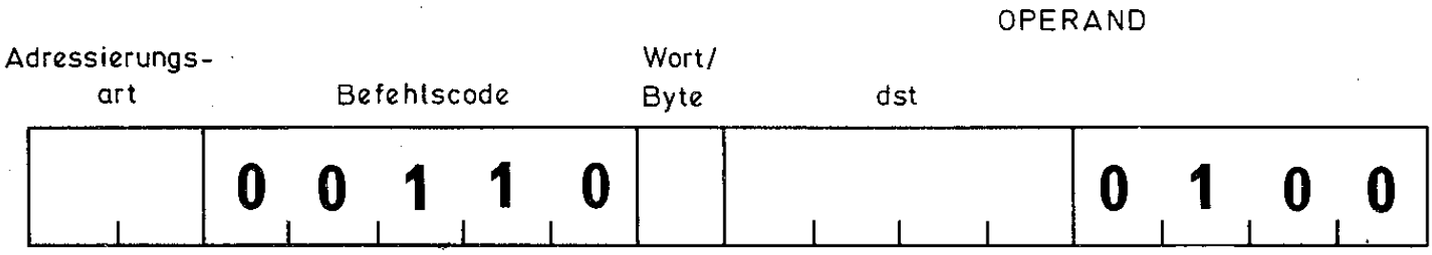
Adressierungsart	dst	src
10	R	IM
00	IR	IM (dst ≠0)
01	DA	IM (dst = 0)
01	X	IM (dst ≠0)



src →0

dst	src
R	R

TSET



S-Flag ←dst (= MSB) = 1

Adressierungsart	dst
10	R
00	IR
01	DA (dst = 0)
01	X (dst ≠0)

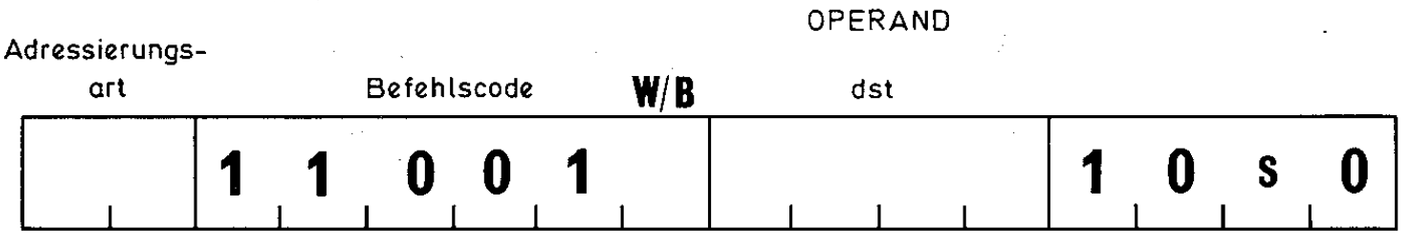
Z-Flag wird gesetzt, wenn das Resultat Null ist; andernfalls wird es zurückgesetzt.

3.4.6 Schiebe- und Rotationsoperationen

Eines der vielen Zeichen der außergewöhnlichen Klasse des Z 8000 sind seine Schiebe- und Rotationsoperationen, die in ihrer Vielfalt grundsätzlich für jedes Allzweckregister anwendbar sind. Es ist praktisch alles möglich, was an Verschiebung von Datenworten nur irgendwie denkbar ist: Byte, Wort, Doppelwort rechts oder links in Carry (dynamisch), arithmetisch mit 1 an MSB nach rechts mit 1 Vorschub in Carry oder mit 0 an LSB nach links; Byte- und Wort-Rotation rechts oder links durch Carry oder in Carry und nicht zuletzt die BCD-Digit-Rotation für eine BCD-Arithmetik. Was die Befehle dieser Gruppe betrifft, ist der Z 8000 für alle anderen Prozessoren ein echter Maßstab. Auf die Funktion der Verschiebeoperationen wird in Kapitel 6 (Abschnitt 1) noch eingegangen. Folgende Tabelle zeigt die Befehle dieser Gruppe aufgelistet:

Befehl	Operanden		Kurzbezeichnung
	Ziel	Quelle	
RL RLB	•	•	Rotation nach links in Carry (C-, Z-, S- und V-Flag betroffen)
RLC RLCB	•	•	Rotation nach links durch Carry (C-, Z-, S- und V-Flag betroffen)
RLDB	•	•	Digit-Rotation nach links (Z- und S-Flag betroffen)
RR RRB	•	•	Rotation nach rechts in Carry (C-, Z-, S- und V-Flag betroffen)
RRC RRCB	•	•	Rotation nach rechts durch Carry (C-, Z-, S- und V-Flag betroffen)
RRDB	•	•	Digit-Rotation nach rechts (Z- und S-Flag betroffen)
SDA SDAB SDAL	•	•	dynamische, arithmetische Verschiebung (links oder rechts in Carry) (C-, Z-, S- und V-Flag betroffen)
SDL SDLB SDLL	•	•	dynamische, logische Verschiebung (links oder rechts) in Carry (C-, Z- und S-Flag betroffen)
SLA SLAB SLAL	•	•	arithmetische Linksverschiebung (Bitposition definierbar) in Carry (C-, Z-, S- und V-Flag betroffen)
SLL SLLB SLLL	•	•	logische Linksverschiebung in Carry (Bitposition definierbar) (C-, Z- und S-Flag betroffen)
SRA SRAB SRAL	•	•	arithmetische Rechtsverschiebung (Bitposition definierbar) in Carry (C-, Z-, S- betroffen, V-Flag wird 0)
SRL SRLB SRLL	•	•	logische Rechtsverschiebung in Carry (Bitposition definierbar) (C-, Z- und S-Flag betroffen)

RL

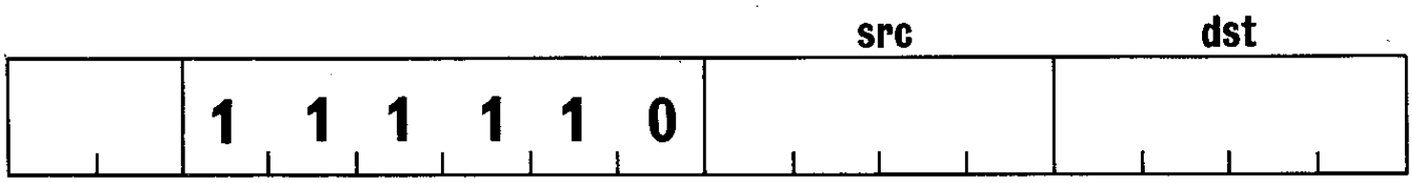


s = 0 einmal
s = 1 zweimal
> links

C ← dst (MSB) = 1
dst (0) ← dst (MSB)
dst (n+1) ← dst (n) (n → 0 . . . MSB – 1)

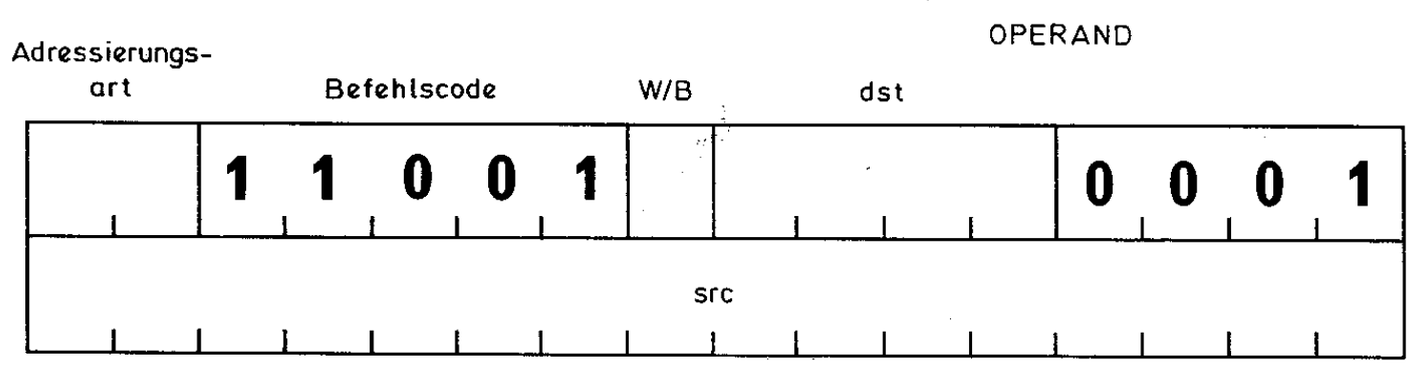
Adressierungsart	dst	src
10	R	IM

RLDB



Adressierungsart	dst	src
10	R	R

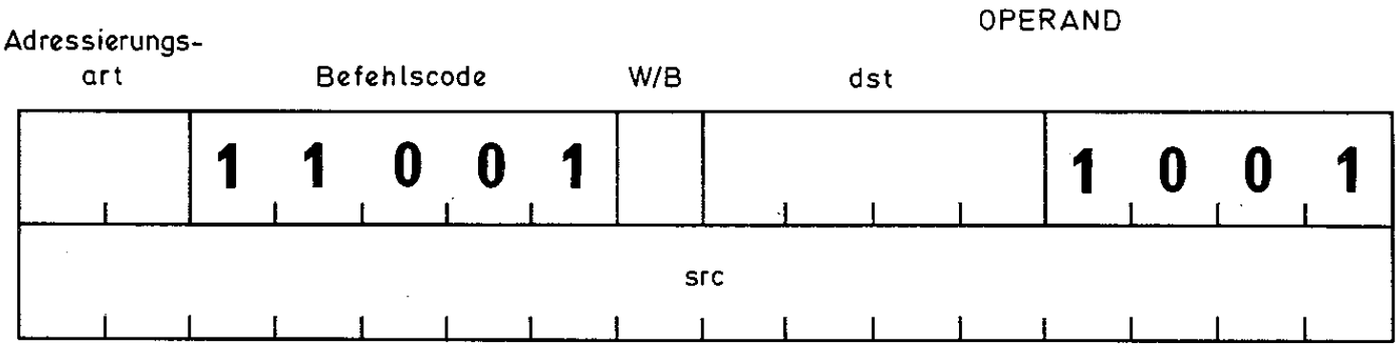
SLL



dst (0) ← 0
dst (n+1) ← dst (n) (n → 0 . . . MSB – 1)
C ← dst (MSB)

Adressierungsart	dst	src
10	R	IM

SRA



dst (MSB) ← dst (MSB)
dst (n) ← dst (n+1) (n → 0 . . . MSB – 1)
C ← dst (0)

Adressierungsart	dst	src
1 0	R	IM

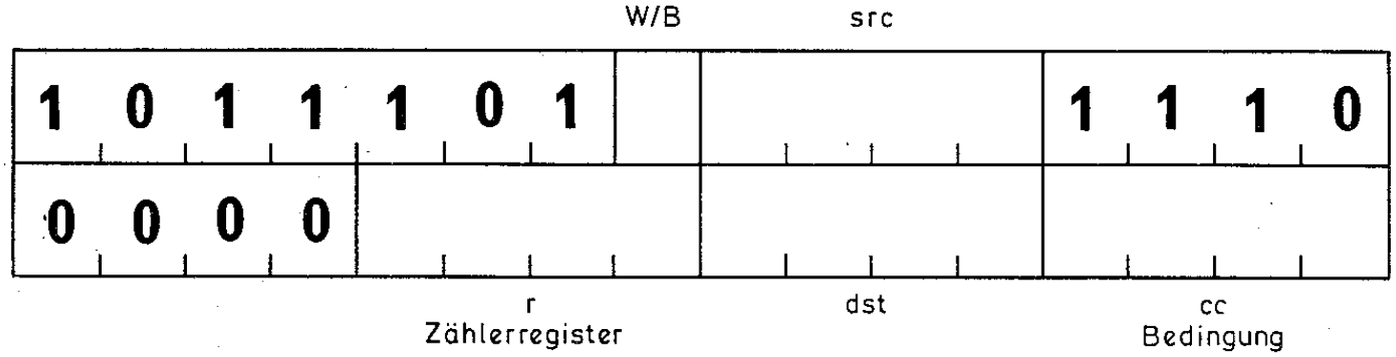
3.4.7 Block-Transfer und String-Operationen

Die Befehle dieser Gruppe gelten als ausgesprochen komplex bezüglich der Prozessoren. Für den Anwender sind sie transparent und einfach anwendbar. 3 Typen bilden vielfältige Varianten: laden, vergleichen und übersetzen (mit Test). Die jeweiligen Typen finden in der wiederholenden Ausführung (jeweils dekrementieren oder inkrementieren) in einem 64 K-Feld ihre höchste Steigerung. Blockdaten-Transfers sind in einem maximal 64 KB umfassenden Datenfeld mit einem einzigen Befehl auszudrücken. Das ist absolut extrem. Speziell sind das die Befehle LDDR bzw. LDIR, CPSDR bzw. CPSIR und TRTDRB bzw. TRTIRB. Bei LDIR gilt ein Register als Index für die Anzahl der zu übertragenden Strings. Zwei weitere Register beinhalten jeweils die Quell- und die Zieladresse des zu übertragenden String (beide 16 bit). Wurde ein Transfer ausgeführt, wird jeweils Quelle und Ziel inkrementiert (um 1 bei Bytes, um 2 Stellen bei LDIR) und das Indexregister um 1 dekrementiert. Bei CPSIR wird prinzipiell in der gleichen Weise verfahren. Die Quell- und Zieladresse dient jedoch im Gegensatz dazu für zwei Operanden, die hintereinander geholt und in der ALU verglichen werden (subtrahiert), wobei das Z-Flag betroffen ist. Nach der Bedingungstabelle in Abschnitt 3.3 wird verglichen. Bei TRTIRB werden zwei Quelladreßregister verwendet, von denen das eine als Indexzeiger auf die jeweilige Stelle in der Tabelle deutet und das andere auf den untersten Wert der Tabelle zeigt. Ein Wert als Adresse aus der Tabelle selbst bildet den Bezug einer Übersetzung der Tabelle. Der Befehl dient dazu, wie schon einmal genannt, Übersetzungen verschiedener Codes auszuführen, beispielsweise den 7-bit-ASCII-Code in einen 8-bit-Code einer bestimmten Parität. Die repetierenden Befehle sind selbstverständlich unterbrechbar. Das ist auch notwendig, denn bei der Bearbeitung eines 64 K-Blocks, bzw. entsprechend langen Tabelle, arbeitet die CPU etwa eine Sekunde. Das ist fast ein "Lichtjahr" bezüglich der Erfordernisse eines dringenden $\overline{\text{NMI}}$. Folgende Tabelle zeigt die Liste der Transfer-Befehle, die, wie schon gesagt, als extrem zu bezeichnen sind:

Befehl	Operanden Ziel	Quelle		Kurzbezeichnung
CPD CPDB	•	•	r, cc	Vergleich String dekrement Zeiger (Z- und V-Flag betroffen), Indexzähler
CPDR CPDRB	•	•	r, cc	Vergleich, dekrementierend, wiederholend, (Z- und V-Flag betroffen)
CPI CPIB	•	•	r, cc	Vergleich String inkrement Zeiger (Z- und V-Flag betroffen), Indexzähler
CPIR CPIRB	•	•	r, cc	Vergleich, inkrementierend, wiederholend (Z- und V-Flag betroffen)
CPSD CPSDB	•	•	r, cc	Vergleich zweier Strings, dekrement (Z- und V-Flag betroffen), Indexzähler
CPSDR CPSDRB	•	•	r, cc	Vergleich zweier Strings, wiederholend, dekrem., (Z-, V-Flag), Indexzähler
CPSI CPSIB	•	•	r, cc	Vergleich zweier Strings, inkrement (Z-, V-Flag), Indexzähler
CPSIR CPSIRB	•	•	r, cc	Vergleich zweier Strings, wiederholend, inkrem., (Z-, V-Flag), Indexzähler
LDD LDDB	•	•	r	laden mit dekrem. Quelle, Ziel (V-Flag betroffen)
LDDR LDDRB	•	•	r	laden mit dekrem. Quelle, Ziel wiederholend (V-Flag betroffen)
LDI LDIB	•	•	r	laden mit inkrem. Quelle, Ziel (V-Flag betroffen)
LDIR LDIRB	•	•	r	laden mit inkrem. Quelle, Ziel wiederholend (V-Flag betroffen)
TRDB	•	•	r	übersetzen dekrem. Ziel, Indexzähler (V-Flag betroffen)
TRDRB	•	•	r	übersetzen dekrem. Ziel, Indexzähler, wiederholend (V-Flag betroffen)
TRIB	•	•	r	übersetzen inkrem. Ziel, Indexzähler (V-Flag betroffen)
TRIRB	•	•	r	übersetzen inkrem. Ziel, Indexzähler, wiederholend (V-Flag betroffen)
TRTDB	src1	src2,r		übersetzen (Test) dekrem. Ziel (Z- und V-Flag betroffen), Indexzähler
TRTDRB	src1	src2,r		übersetzen (Test) dekrem., wiederholend (Z- und V-Flag betroffen), Indexzähler
TRTIB	src1	src2,r		übersetzen (Test) inkrem. Ziel (Z- und V-Flag betroffen), Indexzähler
TRTIRB	src1	src2,r		übersetzen (Test) inkrem., wiederholend (Z- und V-Flag betroffen), Indexzähler

Beispiele einzelner Assembler-Befehle:

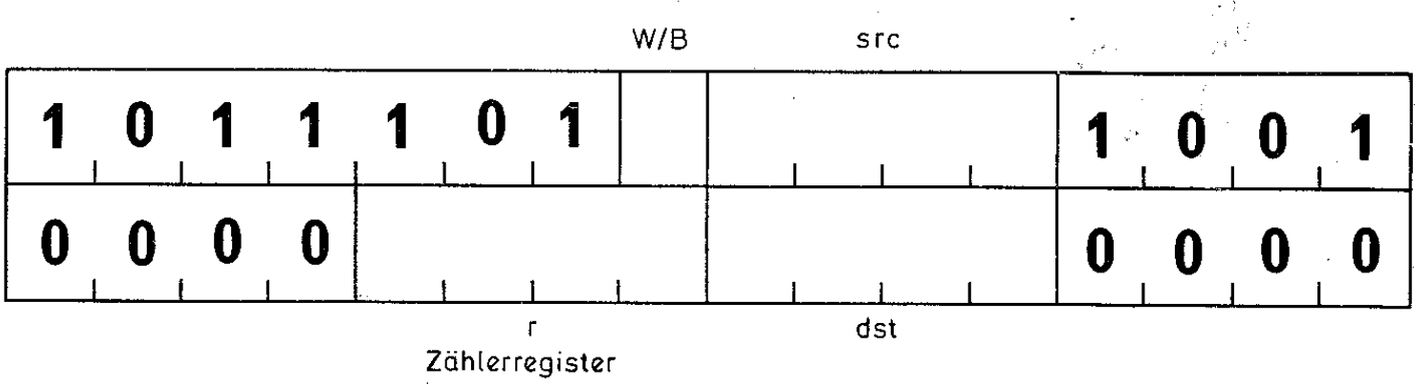
CPSDR



automatisch dekrementieren,
dst und src (—1 bei Byte, —2 bei Wort)
wiederholen, bis cc erfüllt oder r = 0

dst	src	r
IR	IR	R

LDDR

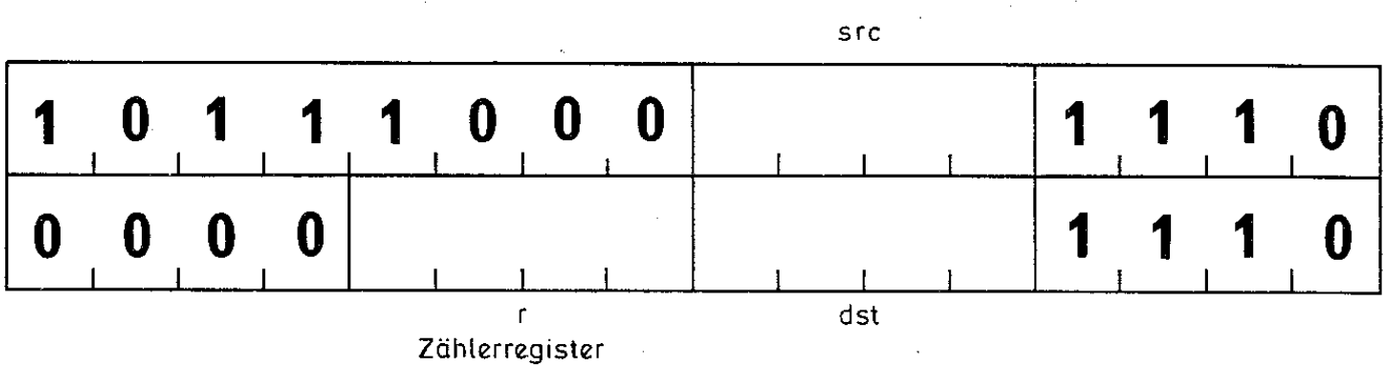


automatisch dekrementieren,
dst und src (—1 bei Byte, —2 bei Wort)
wiederholen, bis r = 0

dst	src	r
IR	IR	R

TRTDRB

RH1 (= 0) ← src 2 [src 1]
automatisches dekrementieren
von src 1 um 1
r (Wort) ← r — 1
wiederholend bis RH1 ≠ 0
oder r = 0



3.4.8 Input/Output-Operationen

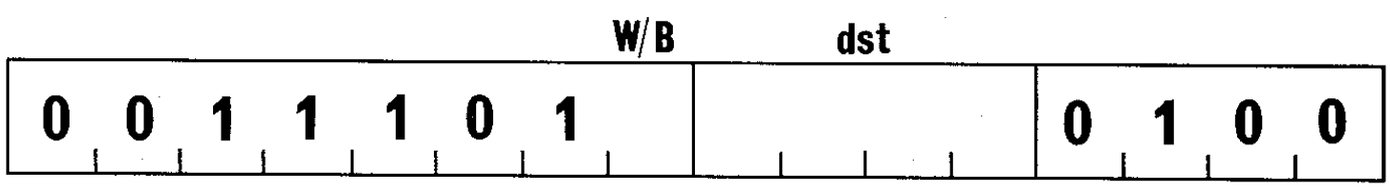
Die Gruppe der Input/Output- oder E/A-Befehle bilden einen gesamten Block privilegierter Operationen. Sie unterteilt sich in allgemeine E/A-Operationen, die die diversen Anwenderschnittstellen eines Systems betreffen, und in die Untergruppe der speziellen E/A-Befehle. Die letzteren betreffen alle Operationen der CPU mit der Z 8010-MMU. Befinden sich zwei und mehr MMUs in einem Z 8001-System, wird man sehr schnell erkennen, daß die relativ große Zahl spezieller MMU-E/A-Befehle keinesfalls übertrieben ist. Eine MMU benötigt für eine Initialisierung etwa 270 Byte, zwei folglich 540 Byte usw. Die CPU ermöglicht einen Transfer von Daten zur MMU während einer Initialisierungsroutine im repetierenden Betrieb, so daß solche Routinen leicht überschaubar geschrieben werden können. Die allgemeinen und speziellen E/A-Befehle (unterschiedliche Anzeige am Status) sind sinngemäß identisch aufgebaut. Die Übernahme erfolgt in Bytes oder Worten, mit Dekrement Indexzähler, Quelladresse fest, Zieladresse nach jedem Transfer dekrementierend oder inkrementierend, in wiederholender Ausführung bis r = 0. Die Quelladresse ist als die eigentliche Portadresse aufzufassen. Die Zieladresse zeigt in den Speicher. Bei der Ausgabe wird bezüglich der Veränderung die Quelladresse betroffen (Zeiger in Speicher). Die Struktur dieser Befehle ist sehr logisch und transparent. Sie sind zu jeder Zeit unterbrechbar und verändern keine Flags (ausgenommen vom V-Flag, wenn der Indexzähler null wurde). Die repetierenden Ausführungen der Befehle haben DMA-Charakter. Folgende Tabelle zeigt die Auflistung dieser Gruppe:

Befehl	Operanden Ziel	Operanden Quelle	Kurzbezeichnung
IN INB	•	•	Übernahme von Port
IND INDB	•	• r	Übernahme von Port mit Dekrement, Indexzähler dekrem.
INDR INDRB	•	• r	Übernahme von Port mit Dekrement, Indexzähler, wiederholend
INI INIB	•	• r	Übernahme von Port mit Inkrement, Indexzähler dekrem.
INIR INIRB	•	• r	Übernahme von Port mit Inkrement, Indexzähler dekrem., wiederholend
OTDR OTDRB	•	• r	Ausgabe an Port mit Dekrement, Indexzähler dekrem., wiederholend
OTIR OTIRB	•	• r	Ausgabe an Port mit Inkrement, Indexzähler dekrem., wiederholend
OUT OUTB	•	•	Ausgabe an Port
OUTD OUTDB	•	• r	Ausgabe an Port mit Dekrement, Indexzähler dekrement
OUTI OUTIB	•	• r	Ausgabe an Port mit Inkrement, Indexzähler dekrement
SIN SINB	•	•	Übernahme von MMU

SIND SINDB	•	• r	Übernahme von MMU mit Dekrement, Index dekrement
SINDR SINDRB	•	• r	Übernahme von MMU mit Dekrement, Index dekrem., wiederholend
SINI SINIB	•	• r	Übernahme von MMU mit Inkrement, Index dekrement
SINIR SINIRB	•	• r	Übernahme von MMU mit Inkrement, Index dekrem., wiederholend
SOTDR SOTDRB	•	• r	Ausgabe an MMU mit Dekrement, Index dekrement, wiederholend
SOTIR SOTIRB	•	• r	Ausgabe an MMU mit Inkrement, Index dekrement, wiederholend
SOUT SOUTB	•	•	Ausgabe an MMU
SOUTD SOUTDB	•	• r	Ausgabe an MMU mit Dekrement, Index dekrement
SOUTI SOUTIB	•	• r	Ausgabe an MMU mit Inkrement, Index dekrement

Beispiele einzelner Assembler-Befehle:

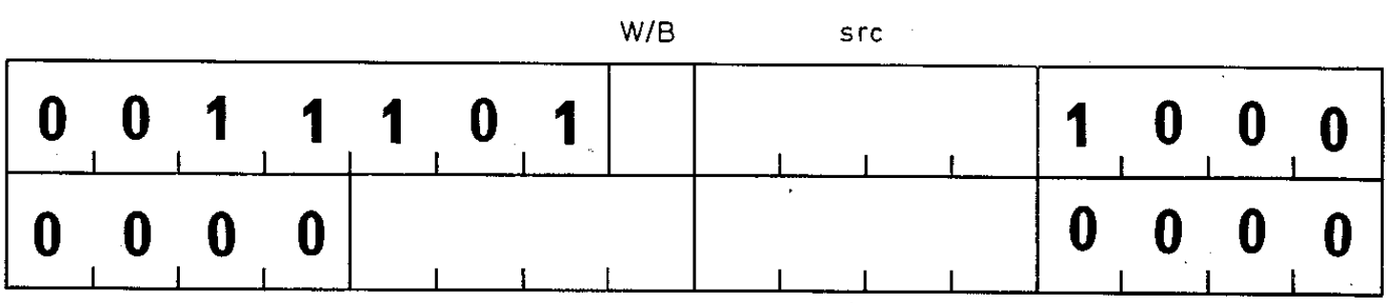
IN



dst ← src

dst	src
R	DA

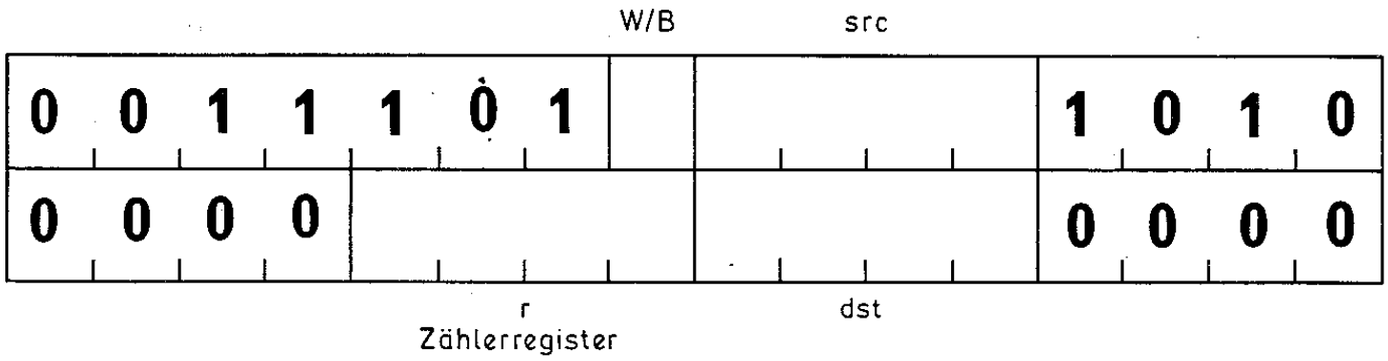
INDR



dst ← src
Zählerregister
automatisches Dekrementieren
von dst (—1 bei Byte, —2 bei Wort)
r ← r — 1 wiederholend bis r = 0

dst	src	r
IR	IR	R

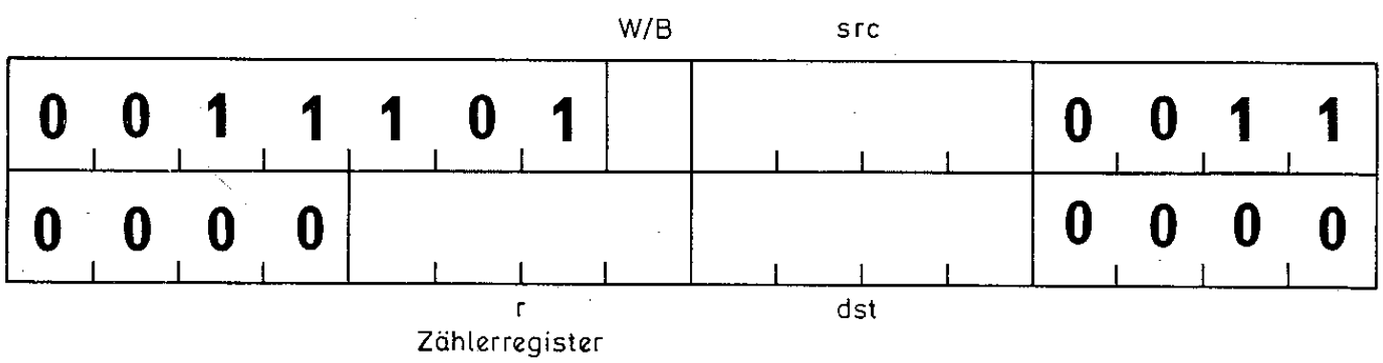
OTDR



dst ← src
automatisches Dekrementieren
von src (–1 bei Byte, –2 bei Wort)
r ← r – 1 wiederholend bis r = 0

dst	src	r
IR	IR	R

SOTIR



dst ← src
automatisches Inkrementieren
von src (+1 bei Byte, +2 bei Wort)
r ← r – 1 wiederholend bis r = 0

dst	src	r
IR	IR	R

3.4.9 CPU-Steueroperationen

Die Befehle dieser Gruppe beziehen sich insbesondere auf das Laden interner CPU-Register, das Setzen und Rücksetzen von Flags, den Programmverzögerungs-Befehl NOP, CPU-HALT und die Multiprozessor-Befehle. Ausgenommen der Multi-Mikro-Befehle handelt es sich um grundsätzliche Operationen, die einen definierten Anfangszustand der CPU sichern. In irgendeiner Form müssen der eine oder andere Befehl in jedem Programm zur Anwendung kommen. Die Auflistung der überwiegend privilegierten CPU-Steuerbefehle zeigt folgende Tabelle:

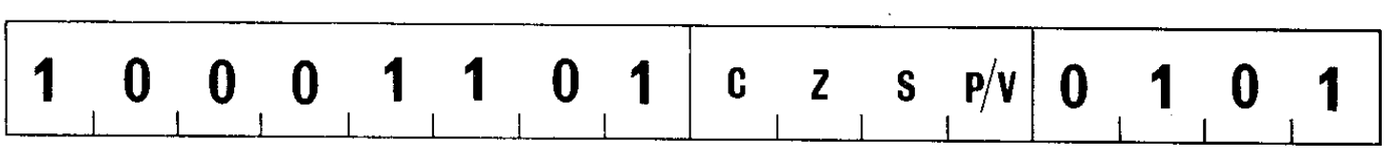
Befehl	Operanden		Kurzbezeichnung
	Ziel	Quelle	
COMFLG	Flag		Komplementbildung der Flags C, Z, S, P/V

● DI	Interrupt		Inaktivieren von VI und/oder NVI im FCW
● EI	Interrupt		Aktivieren von VI und/oder NVI im FCW
● HALT			CPU-HALT, keine Operation, warten auf Int.
LDCTL LDCTLB	●	●	laden – nur Flags C, Z, S, P/V, D, H – oder lesen
● LDCTL	●	●	CPU-Steuerregister, laden oder lesen
● LDPS	●	●	laden Programmstatus (Programmzähler)
● MBIT			Abfrage des $\overline{\mu I}$-Anschlusses der CPU
● MREQ	●		Test $\overline{\mu I}$, S → 0, $\overline{\mu O}$-Ausgang setzen
● MRES			$\overline{\mu O}$-Ausgang der CPU inaktivieren
● MSET			$\overline{\mu O}$-Ausgang der CPU aktivieren
NOP			keine Operation, Programm-Verzögerung
RESFLG	Flag		angegebene Flags auf null setzen
SETFLG	Flag		angegebene Flags auf eins setzen

● privilegierter Befehl

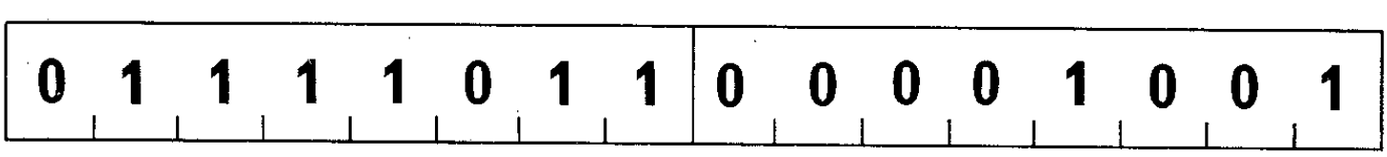
Beispiele einzelner Assembler-Befehle:

COMFLG



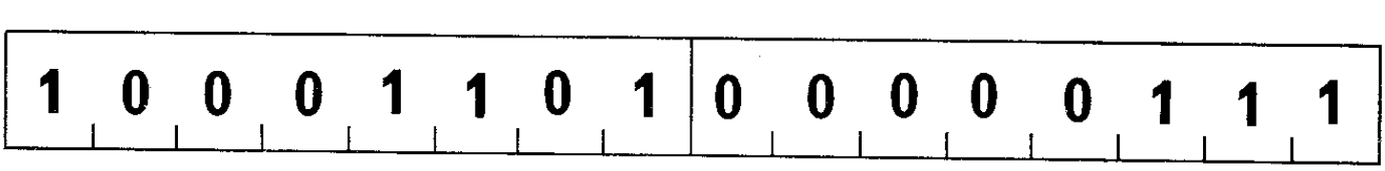
Flags (4 . . . 7) ← Flags XOR Befehl

MRES



$\overline{\mu O}$ ← Null

NOP



Hardware-Konfigurationen für beide Z8000 CPU-Typen

4

4.1 GRUNDSCHALTUNG ZUR BUSTRENNUNG, BETRIEBSZUSTANDS-DEFINITION, TAKTERZEUGUNG, STEUERSIGNALPUFFERUNG UND E/A-ADRESSEPARIERUNG

Bustrennung

Dem gemeinsamen Adreß-/Datenbus der CPU muß man Rechnung tragen. Die interne Zusammenlegung beider Signaltypen muß extern wieder aufgelöst werden. Übliche Speicher- und Schnittstellenbausteine besitzen nicht die Möglichkeit, Adressen und Daten zu unterscheiden. Deshalb ist praktisch in jeder Hardwarekonfiguration mit dem Z 8000 eine Schaltung zur Bustrennung notwendig. Es gibt Ausnahmen einzelner Bausteintypen, die Adressen und Daten auf keinen getrennten Leitungen benötigen. Diese Bausteintypen gehören jedoch ohnehin zur Z-Familie. Darauf wird in einem späteren Kapitel eingegangen.

Um die Trennung von Adresse und Daten wieder herbeizuführen, ist ein Interface (im geringsten Fall) von 4 Bausteinen notwendig. Diese Schaltung hat zum Ziel, auf einen 16-bit-Buspfad einen Adreßwert (Offset) in einen Zwischenspeicher abzulegen, der für die Länge eines Maschinenzyklus gehalten wird. Zu Beginn des zweiten 16-bit-Pfades ist das Datentor für Übertragungen von Eingangs- und Ausgangsdaten zum Speicher oder zu Schnittstellen eingebaut. Wie in Kapitel 2.7 gezeigt wurde, gibt die CPU grundsätzlich zuerst die Adresse aus und sendet bzw. holt darauf ein Datenwort oder -byte. **Bild 4.1** zeigt die schematische Darstellung der globalen Signalaufbereitung für ein Z 8000-Hardwaresystem beliebiger Anwendungen. Das Ziel ist ein getrennter Adreß-/Datenbus. Die Steuerung erfolgt direkt von der CPU mit den Signalen $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ und $\overline{BUSAK}$. Die $\overline{BUSAK}$-Leitung dient zur generellen Freigabe der Tri-State-Treiber. Hat die CPU die Priorität auf den Adreß-/Datenbus, liefert sie immer ein Hi an diesem Ausgang. Hat die CPU eine externe Busanforderung akzeptiert, wird immer Lo an $\overline{BUSAK}$ anstehen, das alle Treiber in einen hochohmigen Zustand versetzt. Man kann $\overline{BUSAK}$ in diesem Sinn als übergeordnetes Signal betrachten. Die echten Steuersignale sind die verbleibenden. Die Lo↑Hi-Flanke von $\overline{AS}$ löst die Zwischenspeicherung der Adresse aus. Sie liegt dann über den Verlauf des Maschinenzyklusses an den Tri-State-Treiberausgängen an. $\overline{DS}$ gleich Lo ist eine Bedingung für die Freigabe des Datentors. R/$\overline{W}$ ist die zweite. Je nach Art des Transfers (schreiben oder lesen) kann der Übertrager nur in einer Richtung durchlässig sein.

Die Bus-Interface-Bausteine sind in zweierlei Art aufgebaut. Die Daten-Übertrager müssen keine Speicherfunktion ausführen. Sie bestehen aus steuerbaren Treibern. Unter steuerbar sind zwei Zustände gemeint. Entweder ist der Treiber durchlässig oder im hochohmigen Zustand. Diese allgemein als Gatter mit drei Zuständen bekannten Logikschaltungen haben im durchlässigen Zustand Hi oder Lo am Ausgang anliegen. Im hochohmigen Zustand ist der Ausgang einerseits inaktiv und andererseits für ein von einem anderen Ausgang produzierten Signal als so hochohmig anzuse-

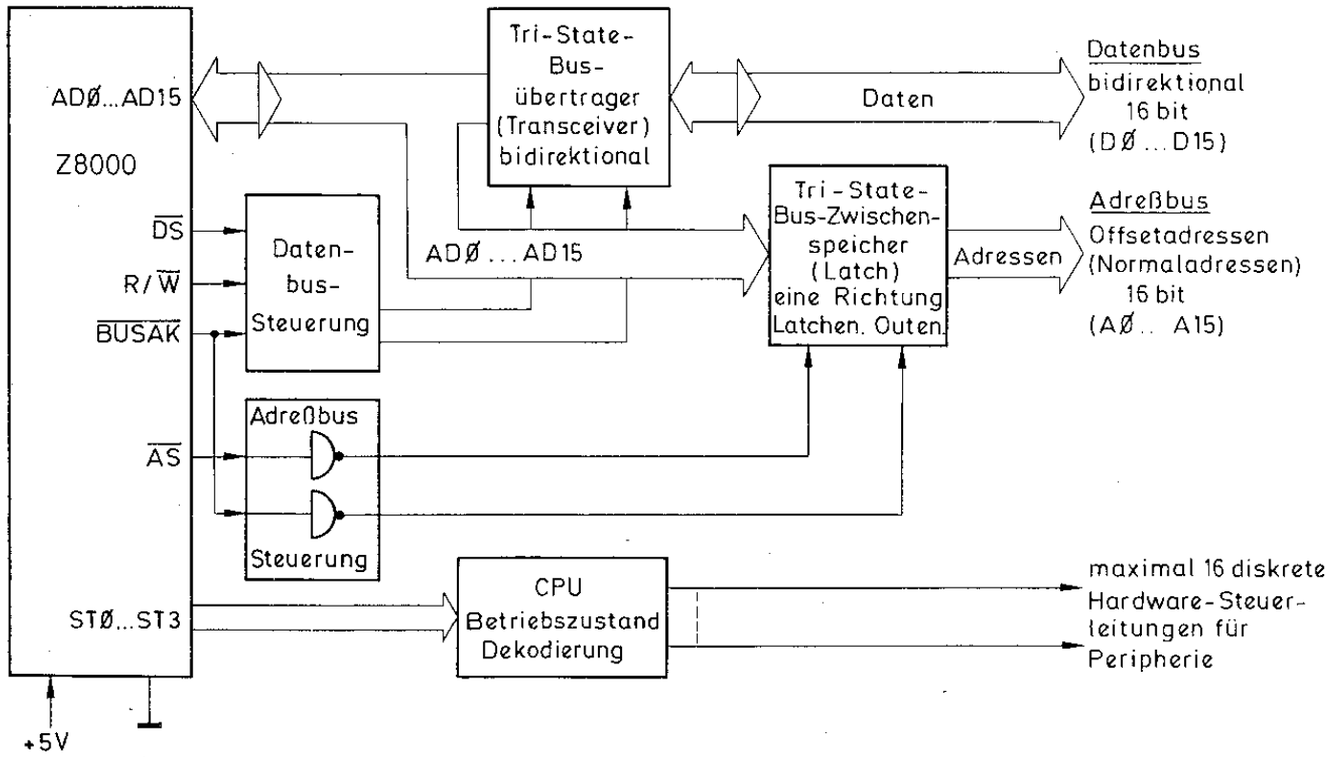


Bild 4.1

hen, daß er nur sehr gering belastend wirkt. Liegt ein Hisan dem im hochohmigen Zustand befindlichen Gatter-Ausgang, fließt nur ein Strom von etwa 0,2 mA (maximal) in diesen Ausgang hinein. Das bedeutet etwa einen Wider-

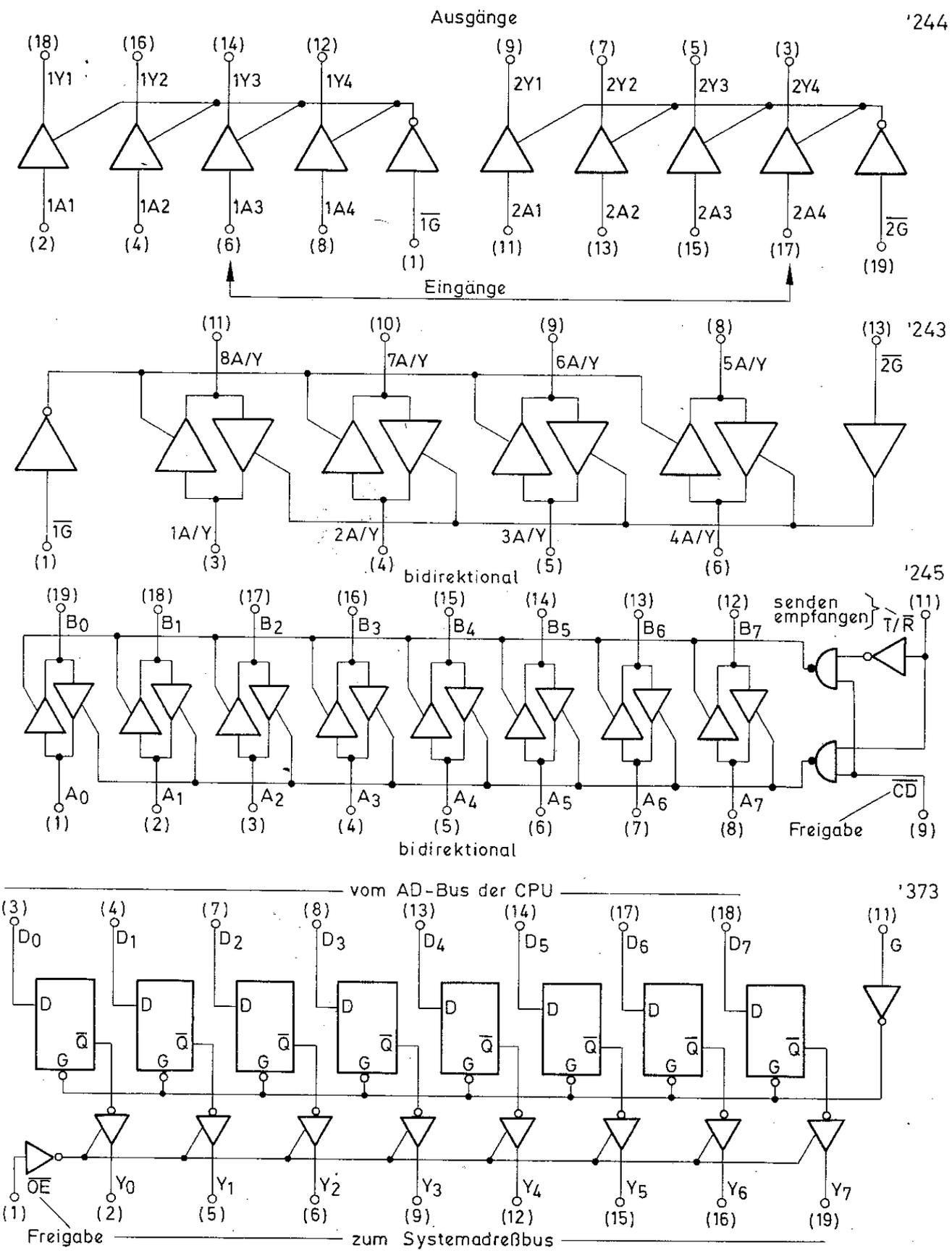

Ausgänge
'244
(18) 1Y1
(16) 1Y2
(14) 1Y3
(12) 1Y4
(9) 2Y1
(7) 2Y2
(5) 2Y3
(3) 2Y4
1A1 (2)
1A2 (4)
1A3 (6)
1A4 (8)
1G (1)
2A1 (11)
2A2 (13)
2A3 (15)
2A4 (17)
2G (19)
Eingänge
(11) 8A/Y
(10) 7A/Y
(9) 6A/Y
(8) 5A/Y
(13) 2G
'243
1G (1)
1A/Y (3)
2A/Y (4)
3A/Y (5)
4A/Y (6)
bidirektional
'245
(19) B0
(18) B1
(17) B2
(16) B3
(15) B4
(14) B5
(13) B6
(12) B7
senden
empfangen
(11) T/R
A0 (1)
A1 (2)
A2 (3)
A3 (4)
A4 (5)
A5 (6)
A6 (7)
A7 (8)
CD (9)
Freigabe
bidirektional
vom AD-Bus der CPU
'373
(3) D0
(4) D1
(7) D2
(8) D3
(13) D4
(14) D5
(17) D6
(18) D7
(11) G
D
Q
G
OE (1)
Y0 (2)
Y1 (5)
Y2 (6)
Y3 (9)
Y4 (12)
Y5 (15)
Y6 (16)
Y7 (19)
Freigabe
zum Systemadreßbus

Bild 4.2

stand von 20 kΩ, mit dem ein Hi (typisch +4 V) belastet wird. 5 Ausgänge im hochohmigen Zustand belasten das Hi erst so hoch als ein Eingang (maximale Belastung 1 mA) eines Gatters. Im aktiven Zustand kann ein Ausgang bis 50 mA absolut sicher liefern (Angaben für AmZ 8104). Diese steuerbaren Treiber-Gatter (meist nicht invertierend) sind überwiegend in oktalen Gruppen in einem Baustein zusammengeschaltet **Bild 4.2** zeigt die drei bekannten Formationen. Der Typ LS 244 kann in einer Richtung oder bidirektional verschaltet werden. 4 Bausteine sind für den Datenbus in diesem Fall nötig. Der Typ 243 ist bereits bidirektional verdrahtet. Jedoch auch hiervon sind 4 Bausteine für den 16-bit-Datenbus nötig. Mit dem 20-poligen Bausteintyp LS 245 erfordert der Aufbau des Datenbus-Interfaces nur noch 2 Bausteine. In einem Baustein sind 16 nicht invertierende Gatter enthalten, wobei jeweils zwei für eine Leitung verknüpft sind. Die dargestellte Nummernbezeichnung der Anschlüsse entspricht der AMD-Variante AmZ 8104.

Für den Adreßbus ist eine Zwischenspeicherfunktion notwendig. In Bild 4.2 unten ist der Innenaufbau des dafür notwendigen Bausteintyps dargestellt. Es handelt sich um ein 8-bit-Zwischenspeicher-Register mit D-Flipflops und nachgeschalteten Tri-State-Treibergattern. Zwei Bausteine sind für den Adreßbus notwendig. Neben dem allgemeinen Typ LS 373 ist der AMD-Baustein AmZ 8173 hierfür einzusetzen.

Bild 4.3 zeigt nun generell eine Grundschaltung eines Z 8000-Basisinterfaces, das für beliebige Ausbaustufen als Standard anzusehen ist. Zwei Bausteine des Typs AmZ 8104 bilden das Interface für die Datenübertragung. Die beiden AmZ 8173 halten an ihren Y-Ausgängen eine Adresse für einen Maschinenzyklus. Die beiden 8-bit-Vergleicher des Typs AmZ 8121 nehmen einen momentanen Adreßwert am Bus über ihre A-Eingänge auf und vergleichen ihn mit einem über DIL-Schalter einstellbaren Zieladreßwert (für E/A-Anwendungen). Mit einem Baustein des Typs Am 25 LS 2537swerden die vier Statusausgänge der CPU in diskrete Einzelsignale dekodiert. Mit oktalen Treibergattern des Typs AmZ 8144 werden alle wichtigen CPU-Signale gepuffert (Status, Steuerleitungen und Segmentadreßleitungen des Z 8001). Die Pufferung dieser Signale ist sicher sehr sinnvoll, wenn beispielsweise der Status für weitere Z 8000-Systembausteine verwendet wird. Als letzter Baustein in diesem Bild sei der vielseitige Zeitgenerator AmZ 8127 genannt. Er faßt eine großes Anzahl verschiedener Funktionen wie CPU-Takt, Wait-Zyklen-Erzeugung, Einzelschrittsteuerung und eine spezielle Auslösung eines NMI zusammen. Die Auslösung des Interrupts erfolgt dann, wenn beispielsweise ein EPROM-Baustein einen Ausfall zeigt.

Die Steuereingänge an den Datenübertragern AmZ 8104 sind CD (Chip sperren) und T/$\overline{R}$ (ausgeben, empfangen). Beim Ausgeben sind die A-Anschlüsse als Eingang und die B-Anschlüsse als Ausgang benutzt. Beim Empfang ist es umgekehrt. Daraus ergibt sich folgende logische Verknüpfung:

Steuer-eingänge	Bedingung		
CD	0	0	1
T/$\overline{R}$	0	1	X*
Busfunktion			
A-Leitungen	Ausg.	Eing.	hochohmig
B-Leitungen	Eing.	Ausg.	hochohmig

* beliebig 0 oder 1

Die Bedingung ist von der CPU leicht zu realisieren. CD muß Lo sein, wenn der Datenbus aktiv sein soll. Das wird damit erreicht, indem $\overline{BUSAK}$ invertiert wird. T/$\overline{R}$ wird aus einer NAND-Verknüpfung mit $\overline{DS}$ und R/$\overline{W}$ angesteuert. Ist DS aktiv, ist es nach dem Invertieren immer Hi. Soll gelesen werden, ist R/$\overline{W}$ gleich Hi, woraus T/$\overline{R}$ Lo wird. Bei T/$\overline{R}$ gleich Losist der Datenbus in Richtung CPU durchlässig. Bei R/$\overline{W}$ gleich Lo in umgekehrter Richtung durchlässig.

Bei den Adressen-Zwischenspeichern

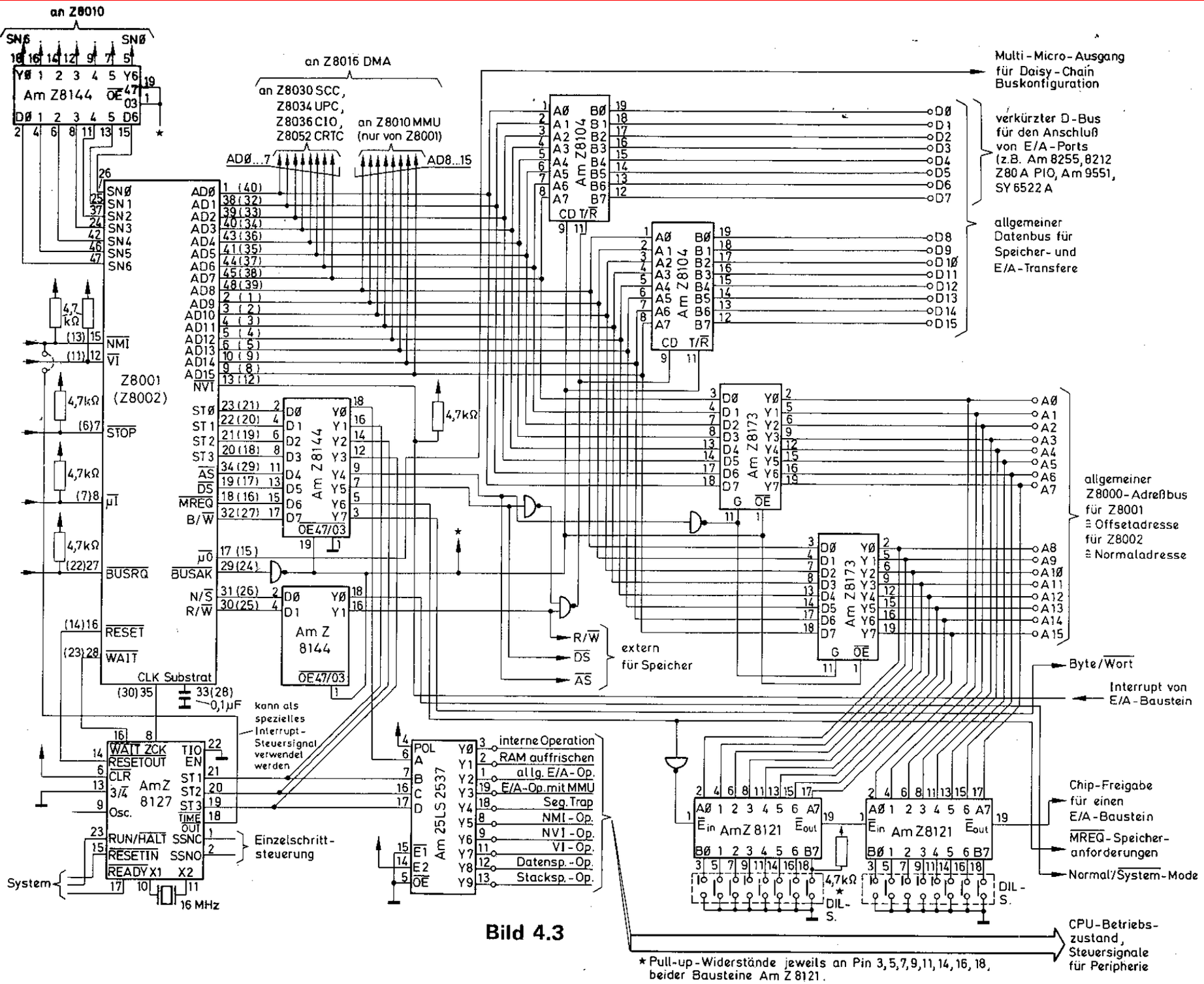


Bild 4.3

★ Pull-up-Widerstände jeweils an Pin 3, 5, 7, 9, 11, 14, 16, 18, beider Bausteine Am Z 8121.

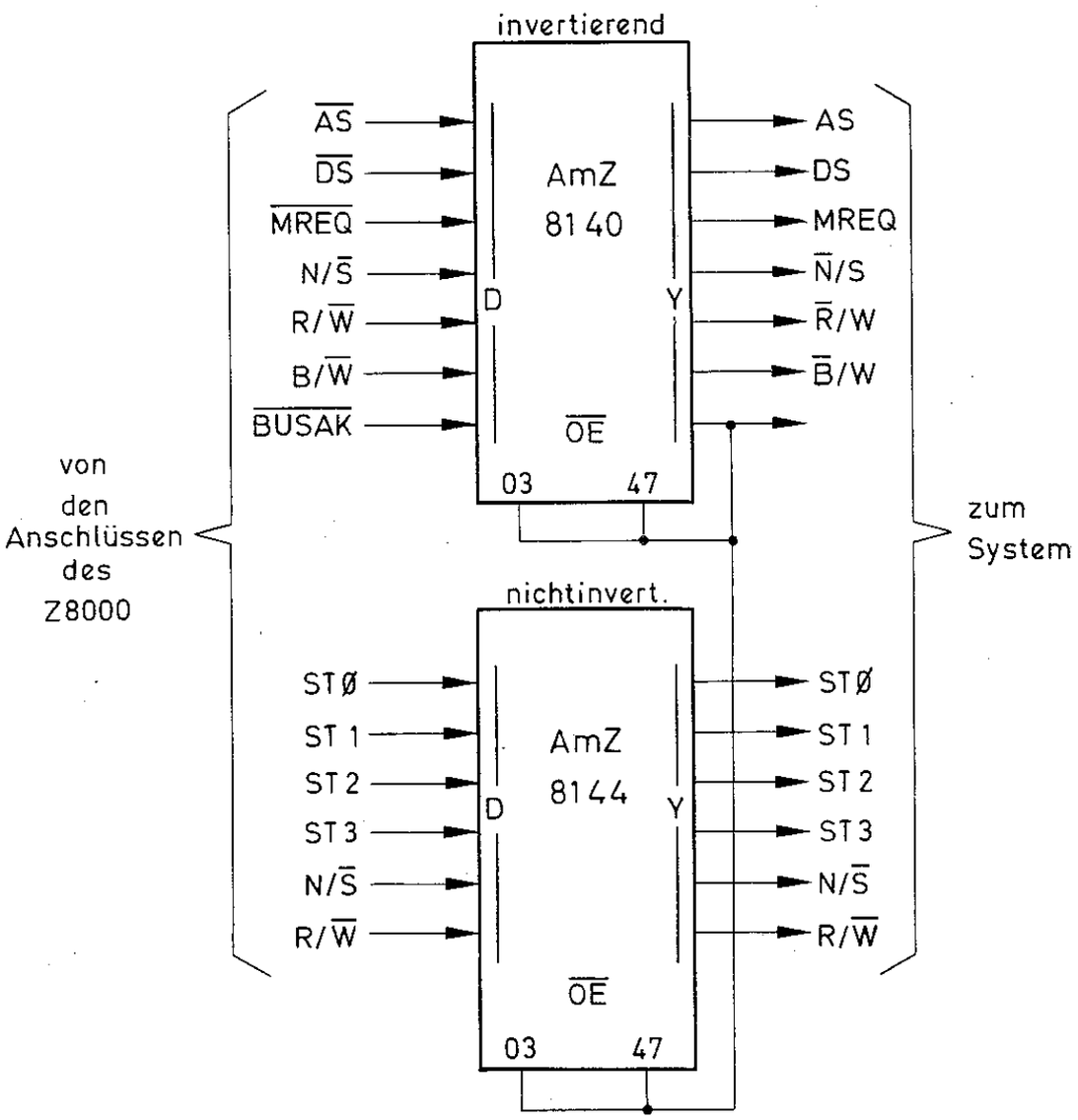


Bild 4.4

AmZ 8173 ist die Steuerung noch einfacher. Das invertierte $\overline{BUSAK}$ dient wieder zur Freigabe der Tri-State-Treiber. Am G-Eingang erfolgt durch den Übergang von Hi auf Lo die Speicherung der an den D-Eingängen anliegenden Werte. Bei diesem Übergang (ausgelöst von dem invertiertem $\overline{AS}$-Signal) liegt an den D-Eingängen eine gültige Adresse an, die im 8173 bis zum nächsten aktivierten $\overline{AS}$ gültig bleibt. Damit ist die Bustrennung in Adreß- und Datenbus abgeschlossen.

Betriebszustands-Anzeige

Die Dekodierung der 4 Statusleitungen wird mit dem Baustein Am 25LS2537 vorgenommen. Dieser Eins-aus-Zehn-Multiplexer erlaubt die Dekodierung der ersten 10 Betriebszustände der CPU. Der Eingang POL legt mit Lo bzw. Hi die Definition eines Ausgang im aktiven Zustand fest. Wird er auf Hi gelegt (wie gezeichnet), sind die Ausgänge Lo-aktiv. Die Ausgänge können ferner durch die Tri-State-Freigabe $\overline{OE}$ nur dann aktiv gemacht werden, wenn das CPU-Signal $\overline{BUSAK}$ gleich Hi ist, der Bus also der CPU gehört. In Bild 4.3 ist das nicht so festgelegt. Man kann es nach Wunsch ändern.

Takterzeugung

In Abschnitt 2.4 wurde in Bild 2.13 eine Schaltung für einen CPU-Takt angege-

ben. Mit dem Baustein AmZ 8127 ist das Taktproblem umfassend und elegant lösbar. Er enthält bereits die beschriebene Gegentakt-Endstufe als Takttreiber, der an ZCK den Takt direkt über CLK der CPU zu überführen gestattet. Dieser Takt ist ebenso für die Z 8010-MMU und den Z 8016-DMATC verwendbar. Der interne Oszillator erzeugt einen TTL-kompatiblen Takt mit einer 16-MHz-Quarz-Synchronisation. Am Eingang 3/$\overline{4}$ wird der interne Takt steuerbar entweder durch 3 oder durch 4 geteilt. Neben dem 4-MHz-Normaltakt ergeben sich 5,33 MHz für die Z 8000A-Version. Ein CPU-Wartezyklus kann über den Eingang RUN/$\overline{\text{HALT}}$ gesteuert werden. Die CPU übernimmt am WAIT-Eingang entsprechende Hardwarekommandos. Der Ausgang $\overline{\text{TIMEOUT}}$ teilt der CPU mit, daß bereits mehr als 16 Wartezyklen überschritten wurden, d.h. der einen WAIT anfordernde Baustein muß defekt sein, weil kein Signal an ihm den WAIT der CPU beendet. $\overline{\text{TIMEOUT}}$ gilt somit als Auslösung eines (NMI-) Interrupts zur softwaremäßigen Behandlung eines solchen Ereignisses.

Steuersignalpufferung

Wie aus Bild 4.3 zu ersehen ist, werden die CPU-Steuersignallausgänge über nicht invertierende Treiber vor äußerer Logik gewissermaßen "abgeschirmt". Der Vorteil liegt generell darin, daß jeder CPU-Ausgang einer konstanten Belastung von 0,1 mA bei Hi unterworfen ist und überdies keine kritische Spannung an den Ausgang gelangen kann, die der CPU Schaden verursacht. Verschiedene andere Eingänge der CPU wie Interrupts oder Busanforderungen werden mit Widerständen statisch in den nicht aktiven Zustand gezogen. Dies sei nur nebenbei bemerkt. Die Steuersignalpufferung ist empfehlenswerter Weise auch nach der Art in **Bild 4.4** aufzubauen. Die Signale $\overline{\text{AS}}$ und $\overline{\text{DS}}$ werden für die Bussteuerung invertiert benötigt. Man kann sich zwei Inverter sparen, die in der Schaltung nach Bild 4.3 erforderlich sind.

E/A-Adreßseparierung

Die 16-bit-Offsetadresse wird zur Adressierung von Speichern oder E/A-Bausteinen benutzt. Der Unterschied zwischen einer Speicher- und einer E/A-Adresse wird einerseits durch das Signal $\overline{\text{MREQ}}$ und andererseits durch die Statusausgabe Nr. 2 gekennzeichnet. In einem Computersystem muß eine separate Adresse oder ein Adreßblock für Schnittstellenoperationen selektiert werden. In der Schaltung in Bild 4.3 sind dafür die beiden Vergleicher AmZ 8121 vorgesehen, wie bereits erwähnt wurde. Eine dieser Gatterschaltungen besteht aus 8 Exklusiv-NOR-Gattern, deren Ausgänge alle auf ein NAND-Gatter geführt werden. Sind die beiden Eingänge der EXNOR-Gatter jeweils gleich belegt (beide Lo oder Hi), ist am Ausgang Hi festzustellen. Haben alle EXNOR-Gatter Hi am Ausgang, ist die erste Bedingung für die Aktivierung von $\overline{E_{OUT}}$ erfüllt. Der NAND-Ausgang entspricht $\overline{E_{OUT}}$. Die zweite Bedingung für die Aktivierung ist der Eingang $\overline{E_{IN}}$, der Lo sein muß. Die neun NAND-Eingänge sind somit alle Hi, woraus $\overline{E_{OUT}}$ gleich Lo und damit aktiviert ist. Die Bedingung für $\overline{E_{IN}}$ wird sehr einfach mit $\overline{\text{MREQ}}$ der CPU definiert. Ist MREQ gleich Hi, zeigt das die Ausgabe einer E/A-Adresse an. $\overline{\text{MREQ}}$ bildet die Basis der Separierung. An $\overline{E_{OUT}}$ des zweiten Bausteins wird durch Lo die Übereinstimmung einer Auswahladresse angezeigt. Die Auswahladresse wird an den B-Eingängen starr verdrahtet oder mit den gezeichneten DIL-Schaltern fest eingestellt. Das Lo an $\overline{E_{OUT}}$ dient als Freigabe oder Anwahl eines Schnittstellenbausteins. **Bild 4.5** zeigt ein Beispiel, wie dieses Signal für einen Einadreß-Port angewendet wird. Als Einadreß-Port dient hier der E/A-Schnittstellenbaustein Am 2950. Dieser Baustein enthält zwei 8-bit-Register aus D-Flipflops, die parallel gegensinnig am Bus angeknüpft sind. Die Ausgänge beider Register sind mit Tri-State-Treibern versehen, sodaß ein E/A-Betrieb möglich ist. Der Baustein verfügt über vielfäl-

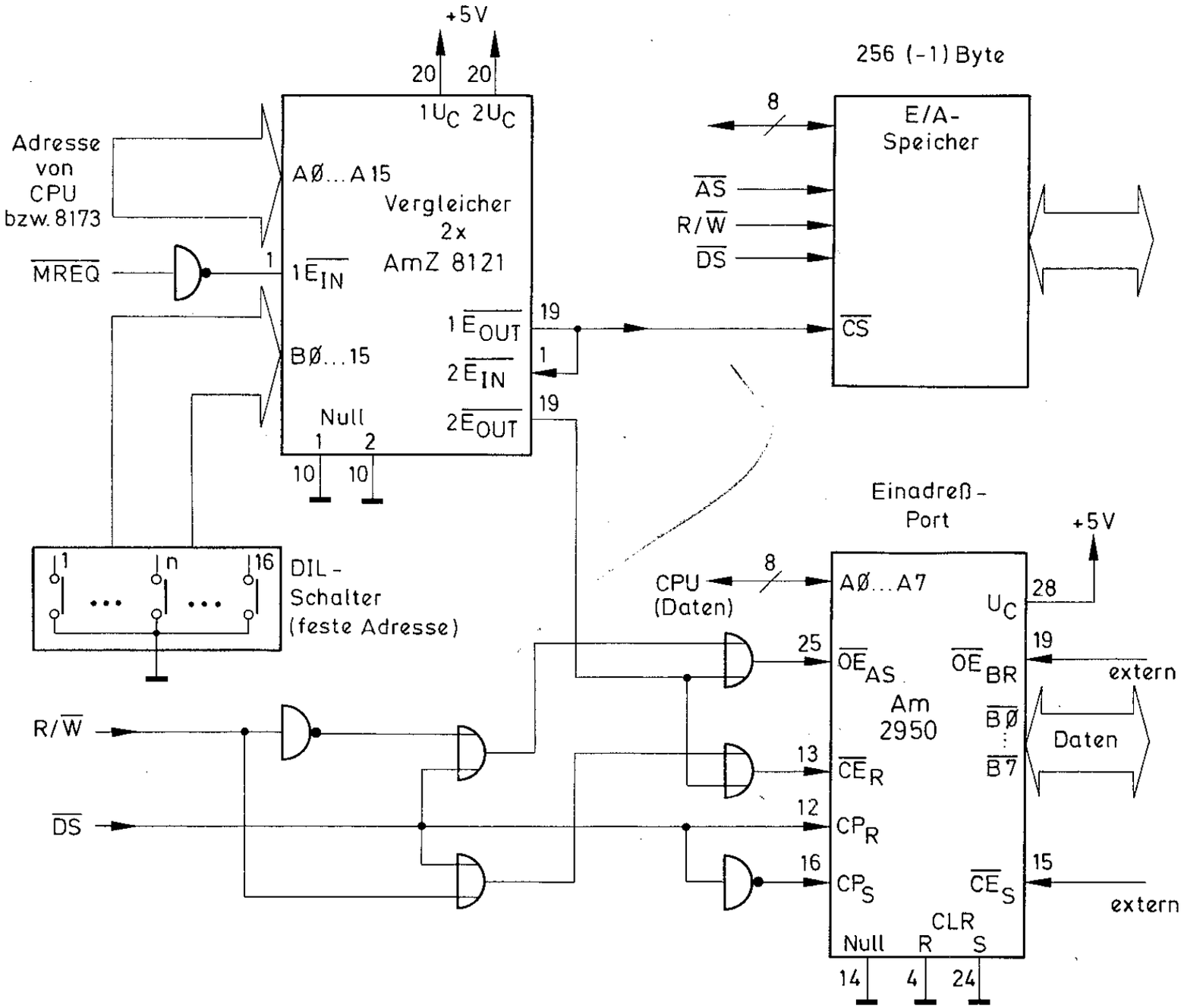

+5V
20
20
1UC 2UC
Adresse von CPU bzw. 8173
AØ...A15
Vergleicher 2x AmZ 8121
MREQ
1
1EIN
1EOUT
19
BØ...15
2EIN
1
Null
2EOUT
19
1
2
10
10
256 (-1) Byte
8
E/A-Speicher
AS
R/W
DS
CS
Einadreß-Port
+5V
DIL-Schalter (feste Adresse)
1
n
16
CPU (Daten)
8
AØ...A7
UC
28
25
OEAS
OEBR
19
extern
Am 2950
BØ
B7
Daten
R/W
13
CER
DS
12
CPR
16
CPS
CES
15
extern
CLR
Null
R
S
14
4
24

Bild 4.5

tige Steuereingänge, die den E/A-Betrieb im Rahmen der Z 8000-E/A-Befehle zulassen. Die beiden internen Register S und R dienen als Zwischenspeicher für Datenworte nach und von draußen. Das S-Register ist mit seinen Ausgängen an die A-Anschlüsse des Bausteins gelegt, zeigt also in Richtung des System-Datenbusses. Das Vergleichssignal 2 $\overline{E_{OUT}}$ des zweiten Adreßvergleichsbausteins wird an den Am 2950 zweifach angebunden. Am Steuereingang $\overline{OE}$AS bewirkt ein Lo die Freigabe der S-Register-Ausgänge. Ist $\overline{OE}$AS aktiv, ist das gleichbedeutend einer Leseoperation der CPU. An $\overline{OE}$AS liegt in der gezeichneten Konfiguration dann Lo an, wenn die CPU den Befehl "IN" ausführt. R/$\overline{W}$ ist Hi, $\overline{DS}$ ist Lo, $\overline{E_{OUT}}$ ist Lo. Dementsprechend wird $\overline{OE}$AS Lo. Über den Busanschluß A 0 . . . A 7 wird am unteren Byte D 0 . . . D 7 des Datenbusses das S-Register ausgelesen. Beim Schreiben in den E/A-Port ist das R-Register Ziel der Operation. Bei der Ausführung des "OUT"-Befehls wird der Steuereingang $\overline{CER}$ von der CPU aktiv gemacht. R/$\overline{W}$ ist Lo, $\overline{DS}$ ist bei der Datenausgabe Lo, $\overline{E_{OUT}}$ ist bei der ausgegebenen Auswahladresse gleich Lo, sodaß $\overline{CER}$ Lo wird. $\overline{CER}$ gibt den Registertakt CPR frei. Bei der nächsten Lo ↑ Hi-Flanke von CPR (gleich $\overline{DS}$) wird das ausgegebene Byte über die A-Anschlüsse in das R-Register darin festgehalten. Das R-Register wird über die B-Anschlüsse nach außen gelesen. Es kann ausgelesen werden, wenn $\overline{OE}$BR extern aktiviert ist. Von der CPU wird die Aktivierung nicht ausgelöst. Eine ebenfalls externe Aktivierung durch eine entsprechende Logik erfährt $\overline{CE}$S. An diesem Eingang wird der Takt für das Schreiben des S-Registers freigegeben. Das S-Register wird mit irgendeinem Byte geladen, das die CPU nur lesen kann, wie vorher beschrieben wurde.

Wenn man jetzt noch einmal zur Bedeutung von 2 $\overline{E_{OUT}}$ zurückblickt, ist nur bei seiner Aktivierung der E/A-Transfer möglich. Der Am 2950-Port hat im Programm eine einzige 16-bit-Adresse zugewiesen bekommen. Wird sie von der CPU ausgegeben, ist die erste Bedingung erfüllt. Gelangt die Adreßausgabe im Zusammenhang mit einem IN- oder OUT-Befehl (oder anderen E/A-Befehlen), ist die zweite Bedingung durch $\overline{MREQ}$ gleich Hi erfüllt. Der Portbaustein gilt als Einadreßport, weil nur eine Adresse eine Freigabe erteilt. Dies ist das Prinzip der E/A-Adreßseparierung.

In Bild 4.5 ist noch eine weitere Möglichkeit angegeben. Nimmt man den Ausgang 1 $\overline{E_{OUT}}$ als Freigabesignal, können damit 256—1 = 255 E/A-Adressen angewählt werden. Eine Adresse ist abzuziehen, weil diese der Port beansprucht. Mit den 255 kann man beispielsweise einen E/A Speicher adressieren.

4.2 ANSCHLUSS-SCHEMATIK FÜR STATISCHE SCHREIB-/ LESESPEICHER

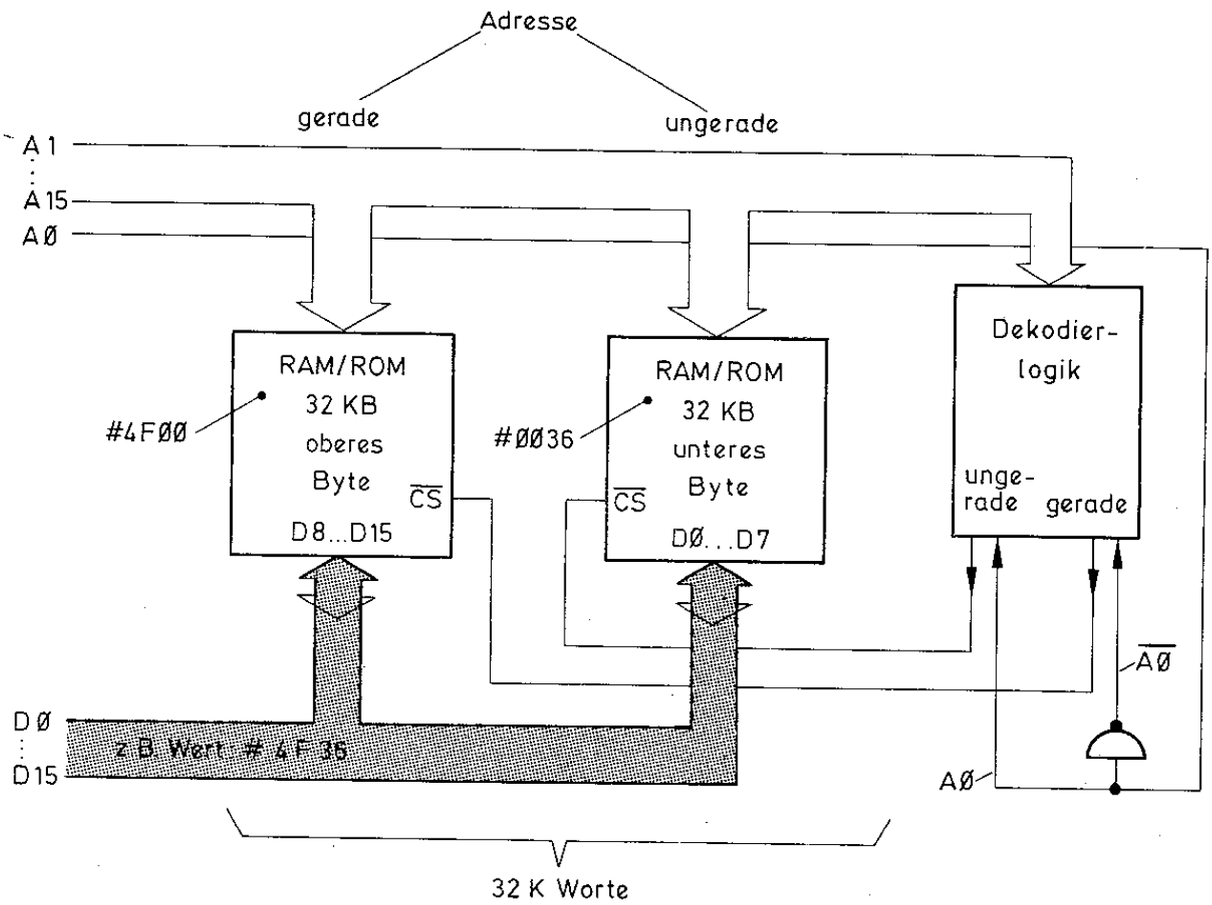
Die separierten Adreß- und Datenleitungen nach Bild 4.3 dienen als Transferleitungen zu beliebigen Halbleiter-Speicherwerken. Man unterscheidet bekanntlich in statische oder dynamische Schreib-/Lesespeicher (RAMs) und in Festwertspeicher wie ROMs, PROMs und EPROMs. Die wichtigsten CPU-Steuersignale für Speicherwerte sind R/$\overline{W}$ (lesen/$\overline{schreiben}$), B/$\overline{W}$ (Byte/$\overline{Wort}$) $\overline{MREQ}$ (Speicherzugriff) und $\overline{DS}$. Bei 16-bit-Prozessoren werden die Speicherwerke im allgemeinen in zwei Zugriffsarten betrieben: Byte- oder Wortzugriff. Warum macht man das? Wie aus den Befehlen des Z 8000 hervorgeht, werden sowohl Byte- als auch Wortoperationen durchgeführt. Arbeitet man mit Bytes (8-bit-Operanden), bliebe beim Schreiben eines Byte in den Speicher immer eine Hälfte des Wortes ungenutzt, würde man das Speicherwerk nur in Worten (16-bit-

Operanden) beschreiben können. Speicherkapazität ist zwar heute bei weitem nicht mehr so teuer als noch vor ein paar Jahren; eine Verschwendung wäre es jedoch trotzdem. Die CPU unterscheidet deshalb selbständig zwischen einem Wort- oder Bytetransfer. Im Grunde ist bei einem Transfer nur die Schreiboperation maßgebend, weil in diesem Fall ein Speicherplatz echt belegt wird. Sie legt fest, ob ein Wort oder ein Byte eingeschrieben wird; ferner genau den Fall betreffend, ob ein Byte in einen Wortspeicherplatz geschrieben wird. Genau diesen Fall will man doch vermeiden. Die CPU zeigt an dem Ausgang B/$\overline{W}$ an, ob sie einen Byte- oder Wortoperanden transferiert. Für die Speicherdekodierlogik ist dieses Signal deshalb maßgebend.

Speicherwerke für den Z 8000 (und auch für andere 16 "bitter") werden grundsätzlich nach dem Schema in **Bild 4.6** aufgebaut. Ein 16-bit-Adreßwert zeigt **immer auf ein Byte.** Die Adresse wird in gerade und ungerade Werte unterschieden. Dies zeigt die Adreßleitung A 0 an, die eine gerade Adresse repräsentiert, wenn A 0 gleich Null ist. In einer Übereinkunft legte man für den Z 8000 fest, daß bei einer **geraden Adresse immer das höherwertige Byte (D8 . . . D15)** des Datenbusses ausgegeben bzw. gelesen wird. Eine **ungerade Adresse (A 0 = 1)** liest bzw. schreibt das **niederwertige Byte** des Datenbusses. Somit steht das Wort 4F36H mit 4F an einer geraden Adresse und 36 an einer ungeraden. Nach diesem Schema ist jedes Speicherwerk bzw. die dazu gehörige Dekodierung zu entwerfen. Die Dekodierung ist praktisch doppelt auszulegen, jeweils für höherwertiges und niederwertiges Byte. Die generelle Freigabe erteilt die Adreßleitung A 0, wenn B/$\overline{W}$ gleich Hi ist. Die größte mögliche Adresse aus 16 bit hat den Wert 65535. Das Byte ist demnach ein niederwertiges, das mit ihrer Hilfe adressiert wird. 64 K Byte sind also in einem Byte orientierten Speicherwerk gleichbedeutend mit 32 K Worte. Der grundsätzliche Zusammenhang in Bild 4.6 gilt für jeden Speichertyp, ob statisch, dynamisch oder Festwert.

Zu Beginn des gesamten Komplexes Speicheranschluß an den Z 8000 wird mit der Dekodierung statischer RAMs begonnen. Wenn auch die Kosten pro Speicherbit zwischen statischen und dynamischen RAMs drastisch zu Gunsten dynamischer Typen zu Buch schlagen und überdies der Z 8000 für den Betrieb dynamischer Speicherwerke sehr gut vorbereitet ist, dürfen die statischen Typen nicht außer Acht gelassen werden. Statisch ist statisch. Es gibt nicht nur extrem schnelle Typen dieser Art (Zugriff $\approx$ 50 ns bei HMOS-Typen); die Datensicherheit liegt bei statischen RAMs noch um eine Größenordnung besser als bei dynamischen Typen. Einen klaren Nachteil verbuchen statische RAMs im Stromverbrauch, der im Schnitt um etwa zwei Größenordnungen pro Bit über dem von dynamischen liegt. Trotzdem, die Verbreitung statischer RAMssist immer noch so hoch, daß sie hier keinesfalls fehlen dürfen.

Bild 4.7 zeigt die komplette Dekodierlogik eines statischen Speicherwerks für Byte- und Wortzugriff mit der Kapazität von 8 K Worte. Die Schaltung enthält zwei getrennte Dekodierbereiche für gerade und ungerade Adressen. Die eigentliche Dekodierung erfolgt durch zwei 3-bit-Binärdekoder/Demultiplexer LS 138, die an ihren diskreten Selektionsausgängen direkt die Freigabeeingänge der Speicherbausteine ansteuern. Eine Selektionsleitung gibt ein Segment von 1 KB frei, gerade wie ungerade. Die Adreßleitungen A1 . . . A10 liegen an allen Bausteinen parallel an. Im gegebenen Fall sind diese Leitungen noch separat zu treiben. Der Datenbus wird über die bereits bekannten bidirektionalen Übertrager AmZ 8104 den Datenein-/ausgängen des jeweiligen Speicherblocks zugeführt. Die Richtungsumschaltung in den Übertragern wird ausschließlich durch die CPU mit R/$\overline{W}$ veranlaßt. Die Freigabe des Speicherwerk-Datenbusses auf den allgemeinen Systemdatenbus wird an $\overline{CE}$ (an CD, Pin 9, des 8104) erteilt. Dies hängt von einigen Bedingungen ab. Zu-


Adresse
gerade
ungerade
A 1
A 15
A Ø
RAM/ROM
32 KB
oberes
Byte
D8...D15
#4FØØ
CS
RAM/ROM
32 KB
unteres
Byte
DØ...D7
#ØØ36
CS
Dekodier-
logik
unge-
rade
gerade
AØ
DØ
D15
z.B. Wort # 4F36
32 K Worte

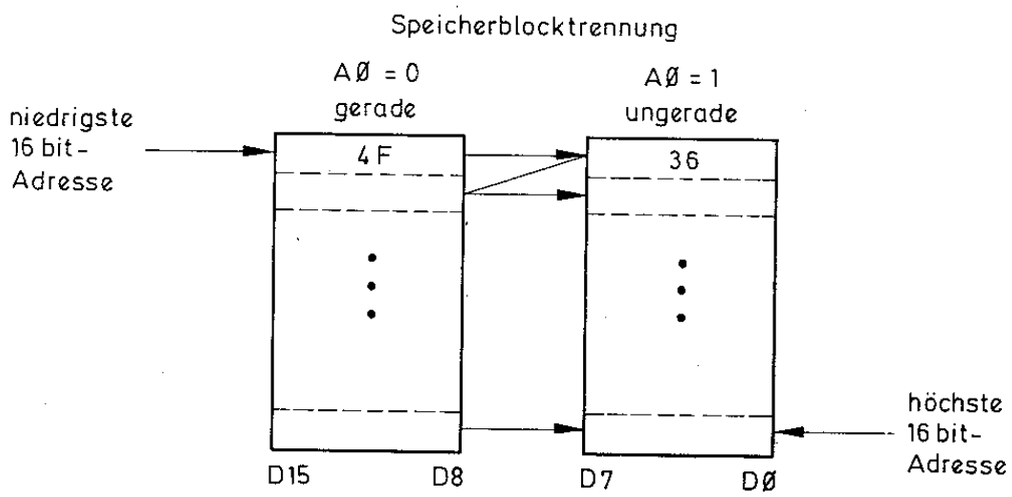

Speicherblocktrennung
AØ = 0
gerade
AØ = 1
ungerade
niedrigste
16 bit-
Adresse
4F
36
höchste
16 bit-
Adresse
D15
D8
D7
DØ

Bild 4.6

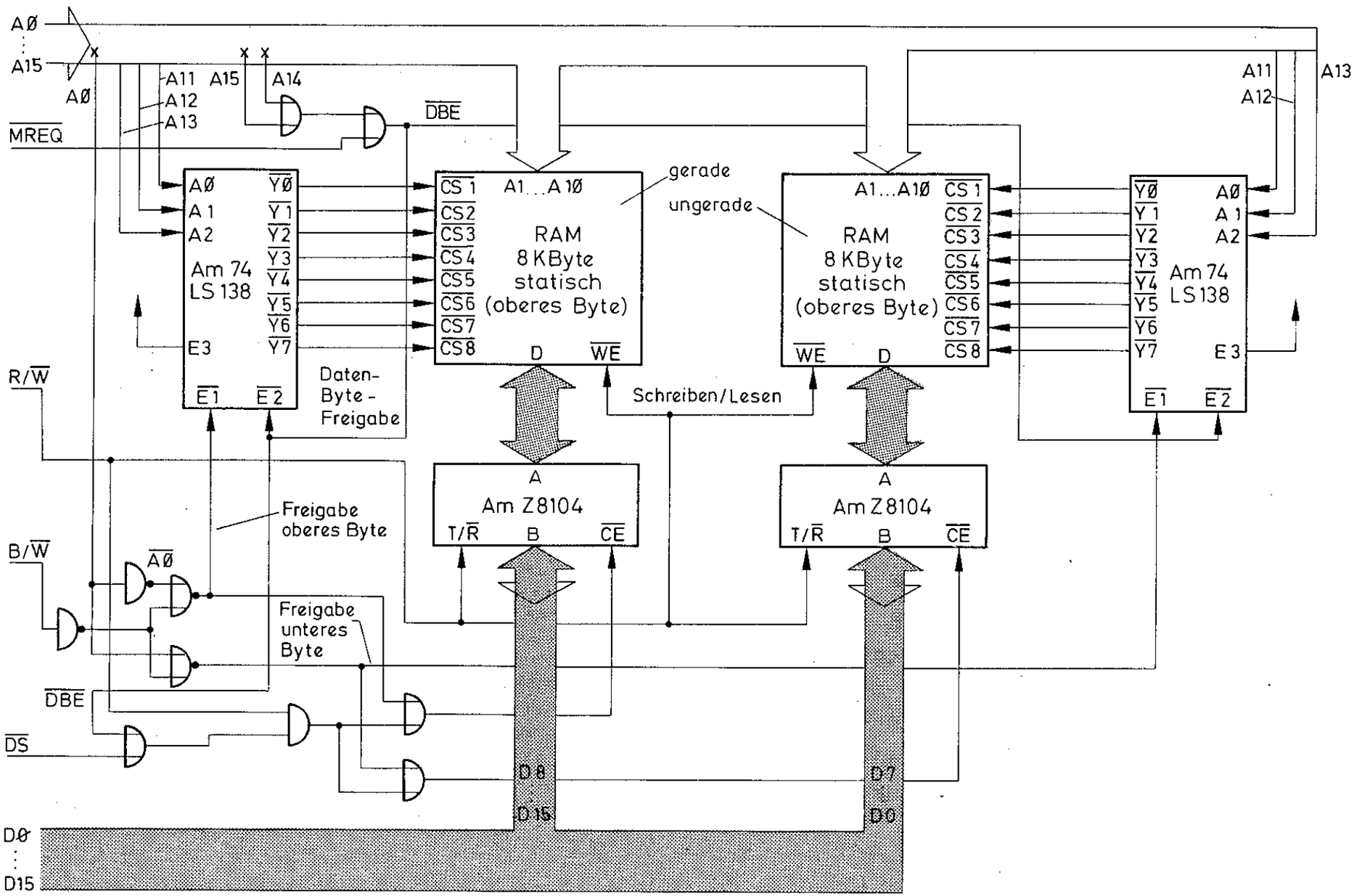

A0
A15
A0
A11
A12
A13
A15
A14
DBE
MREQ
A0
A1
A2
Am 74
LS 138
E3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
E1
E2
Daten-
Byte-
Freigabe
CS1
CS2
CS3
CS4
CS5
CS6
CS7
CS8
A1...A10
RAM
8 KByte
statisch
(oberes Byte)
D
WE
gerade
ungerade
A1...A10
RAM
8 KByte
statisch
(oberes Byte)
WE
D
A11
A12
A13
A0
A1
A2
Am 74
LS138
E3
R/W
Schreiben/Lesen
A
Am Z8104
T/R
B
CE
A
Am Z8104
T/R
B
CE
Freigabe
oberes Byte
B/W
A0
Freigabe
unteres
Byte
DBE
DS
D8
D15
D7
D0
D0
D15

Bild 4.7

erst wird ein allgemeines Freigabesignal mit der Bezeichnung $\overline{DBE}$ erzeugt. Ist dieses Signal aktiv (Lo), muß A14 = A15 = $\overline{MREQ}$ gleich Lo sein. Ist dies erfüllt, wird an den $\overline{E2}$-Eingängen der Binärdekoder diese Bedingung angezeigt. Die zweite Bedingung wird aus der Signalkonfiguration A0 und B/$\overline{W}$ gebildet, wie schon angedeutet. Grundsätzlich werden beide Freigabeeingänge $\overline{E1}$ an den Binärdekodern aktiviert, wenn B/$\overline{W}$ gleich Lo ist, d.h. Wortzugriff. In diesem Fall ist es gleichgültig, welchen Wert A0 hat. Nicht gleichgültig ist der Wert von A0, wenn B/$\overline{W}$ gleich Hi ist. Dann nämlich wird bei A0 = 0 der linke Binärdekoder für das obere Byte freigegeben. Bei A0 = 1 wird der rechte Binärdekoder für das untere Byte freigegeben. Ebenso geschieht das gleichzeitig mit den zugehörigen Übertragern. Der Einfluß von $\overline{DS}$ der CPU bewirkt, daß die Datenbusfreigabe nur zu dem "Augenblick" aktiv ist, worin die CPU selbst durch $\overline{DS}$ gleich Lo anzeigt, daß sie mit Daten am Bus operiert. Das UND-Gatter mit den Eingängen R/$\overline{W}$ und $\overline{DS}$ ∇ $\overline{DBE}$ wird dazu benutzt, um die CD-Eingänge ($\overline{CE}$) beider Übertrager wirklich nur bei einem Datenhol- bzw. -schreibvorgang freizugeben. Bei einem Schreibvorgang wird die Freigabe schon früher erteilt.

In den folgenden Tabellen werden die Bedingungen für einen nochmals zusammengestellt:

allgemeine Speicherfreigabe $\overline{DBE}$ = 0	A 14 0	A 15 0	$\overline{MREQ}$ 0

Wortzugriff B/$\overline{W}$ = 0	$\overline{DBE}$ 0	A 0 beliebig

Bytezugriff B/$\overline{W}$ = 1	$\overline{DBE}$ 0	A 0 = 0 D8 . . . D15	A 0 = 1 D0 . . . D7

schreiben R/$\overline{W}$ = 0	T/$\overline{R}$ = 0 CPU → B → A → Speicher	$\overline{CE}$ = CD 0

lesen R/$\overline{W}$ = 1	T/$\overline{R}$ = 1 CPU ← B ← A ← Speicher	$\overline{CE}$ = CD 0

Wenn man die Leseoperation betrachtet, stellt man fest, daß der Transfer vom Speicherbaustein in A des Übertragers, bei B wieder heraus in Richtung CPU-Bus verläuft. Es muß sichergestellt sein, daß die Übertrager bei B in Bild 4.3 den Eingang haben. Dies ist gemäß der dargestellten Signale auch gesichert. Zum Wortzugriff ist anzumerken, daß ein Wort nur in einen Speicherplatz mit einer geraden Adresse geschrieben wird. Das sollte jedoch keine absolute Bedingung sein.

4.2.1 Beispiel eines Speicherwerks mit den RAM-Bausteinen 2114 bzw. Am 9124

In Bild 4.7 ist mit den 10 Adreßleitungen zu jeweils einem Speicherblock die Länge von 1K definiert, die ein Baustein haben muß. Man könnte den Typ 2102A mit der Kapazität 1K x 1 Bit einsetzen. Für 8 KB sind 64 Bausteine, für 8 K Worte sind 128 Bausteine notwendig. Der Typ 2102AL-2 hat zwar die erforderliche Zugriffszeit von 250 ns, verbraucht jedoch bei +5 V 342 mW. 8 K Worte verbrauchen 43,776 W bzw. 8,885 A bei 5 V. Das ist entschieden zu viel, abgesehen von der großen Zahl von Bausteinen.

Für das Speicherwerk in Bild 4.7 sollte als kleinste Zelle mindestens der Typ 2114 mit der Organisation 1K x 4 Bit eingesetzt werden. Für 8 K Worte sind nur noch 32 Bausteine nötig. Um weiter Strom zu sparen, sollte z.B. der Typ Am 9124 verwendet werden. Ist die Chipfreigabe ($\overline{CS}$) inaktiv, benötigt der Baustein 158 mW. 8 K Worte erfordern somit etwa 5 W (1 A) im Ruhebetrieb. Weil im aktiven Zustand nur maximal vier Bausteine angesprochen sind (Leistungsbedarf dann 368 mW) erhöht sich die Gesamtleistung nur unwesentlich. **Bild 4.8** zeigt die Anordnung mit 1K x 4 Typen zu 8 K Byte. Dieses Netzwerk ist zweimal erforderlich; für ungerade und gerade Adressen. Die Leitungen Y0 . . . Y7 sind an einen Binärdekoder anzuschließen. Der Datenbus wird an die A-Eingänge des AmZ 8104 angebunden.

Ein Speicherwerk mit statischen RAMs dieser Art ist mit dem Z 8000 problemlos zu betreiben. Will man ohne zusätzliche Wartezyklen auf dieses Speicherwerk zugreifen, muß man den Typ Am 9124EDC verwenden, der mit 200 ns Zugriffszeit angegeben ist.

4.2.2 Beispiel eines Speicherwerks mit dem Typ MK 4118 (1K x 8 Bit)

Die Speicherhersteller haben längst erkannt, daß Halbleiterspeicher generell als großer Umsatzfaktor anzusehen ist. Der Typ 2114 ist bereits von den Byte orientierten statischen RAMs überholt worden. Der 1K x 8 Bit organisierte Typ MK 4118 der Firma Mostek ist der erste Einzelbaustein in Bytebreite. Für 8 K Worte sind nur noch 16 Bausteine erforderlich. Beachtet man, daß der Baustein nur noch 400 mW im aktiven Zustand und 300 mW im inaktiven Zustand ($\overline{CS}$ gleich Hi) benötigt, wird ein Fortschritt schon im DC-Bereich (Gleichstromversorgung) sichtbar. 8 K Worte Speicherkapazität benötigen 980 mA im aktiven Zustand bei Verwendung des Typs 4118. **Bild 4.9** zeigt ein Netzwerk der Größe 8 K Byte. In Bild 4.7 ist es jeweils für den geraden und ungeraden Block einzusetzen. Auch hier ist die Verbindung wieder sehr einfach durchzuführen. Die Leitungen Y 0 . . . Y 7 sind an den jeweiligen Binärdekoder anzuschließen. Die Datenleitungen werden an die Anschlüsse der Übertrager 8104 auf der A-Seite verbunden. Die R/$\overline{W}$-Leitung wird an $\overline{WE}$ der Speicher angeknüpft, wie gezeigt. Ein gewisser Unterschied ergibt sich bei den $\overline{CS}$-Freigabeanschlüssen, die in der Funktion keinesfalls mit $\overline{OE}$ gleichzusetzen sind. $\overline{OE}$ löst bei einer Aktivierung die Datenausgangs-Tri-State-Treiber. Daraus folgt, daß diese nur beim Lesen freigegeben werden dürfen. Man muß wohl oder übel $\overline{OE}$ mit R/$\overline{W}$ verknüpfen, sodaß bei R/$\overline{W}$ gleich Hi nur dann $\overline{OE}$ Lo werden kann. $\overline{OE}$ kann nur dann aktiviert werden, wenn $\overline{CS}$ Lo ist.

Der 4118 ist in seinem internen Interface etwas komplizierter als der 2114. Die Besonderheit des 4118 zeigt sich darin, daß er über einen internen Adressenzwischenspeicher verfügt (Funktion vergleichbar mit AmZ 8173). Das Halten einer Adresse wird über den Eingang $\overline{L}$ am 4118 ausgelöst. Ist $\overline{L}$ gleich Lo,

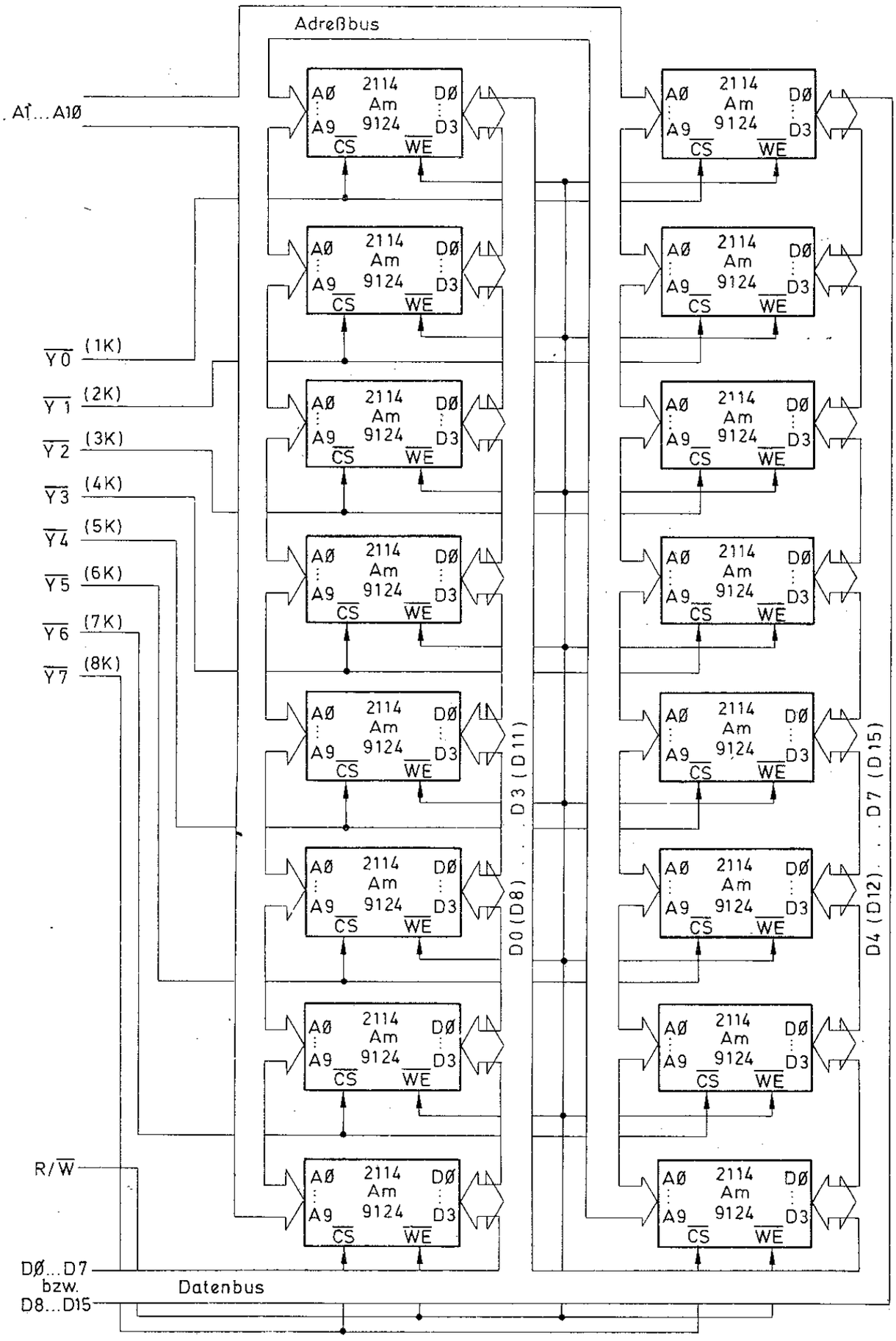

Adreßbus
A1 ... A10
Y0 (1K)
Y1 (2K)
Y2 (3K)
Y3 (4K)
Y4 (5K)
Y5 (6K)
Y6 (7K)
Y7 (8K)
R/W
D0 ... D7
bzw.
D8 ... D15
Datenbus
2114
Am
9124
A0
A9
CS
WE
D0
D3
D0 (D8) . . . D3 (D11)
D4 (D12) . . . D7 (D15)

Bild 4.8

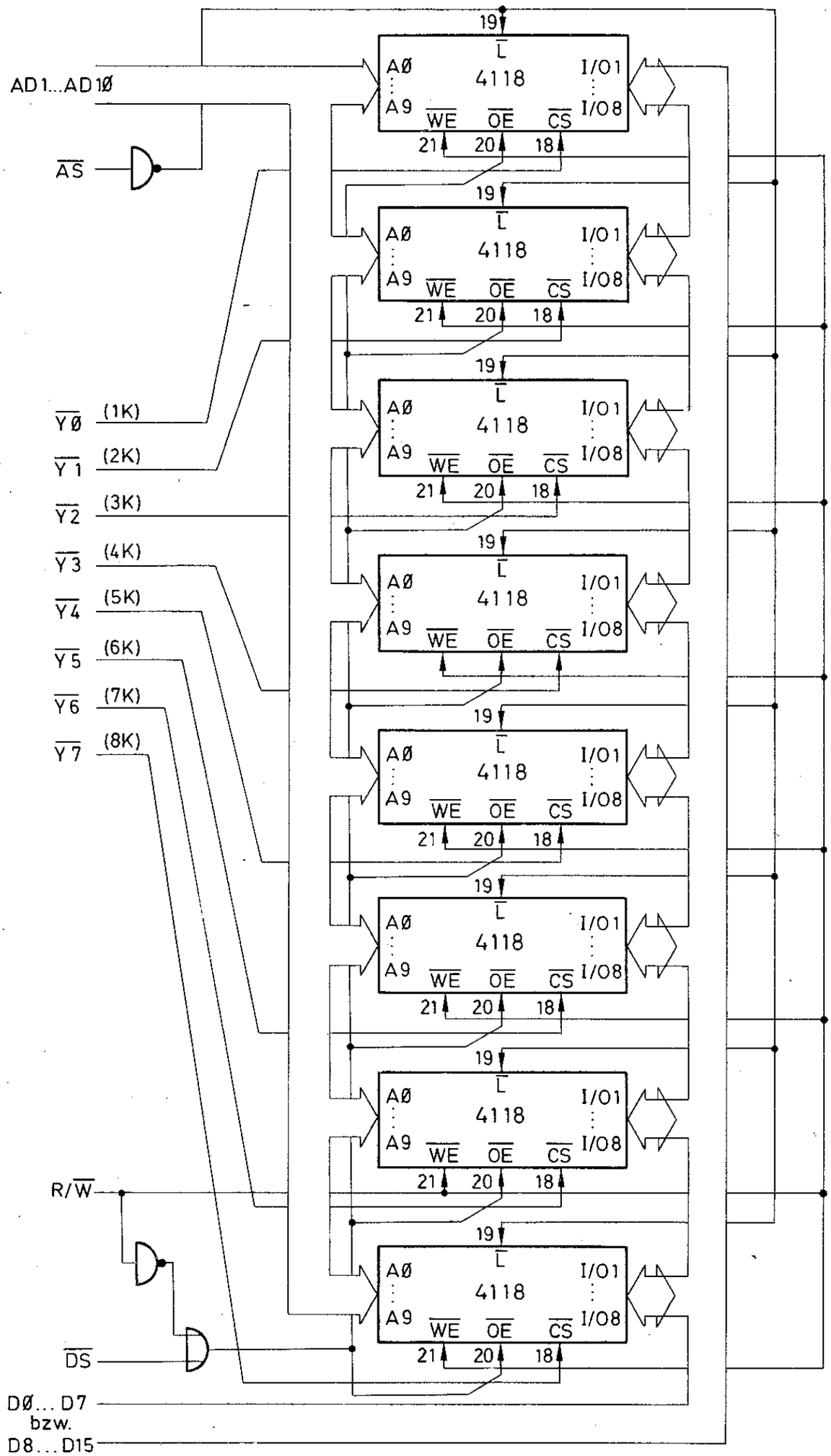
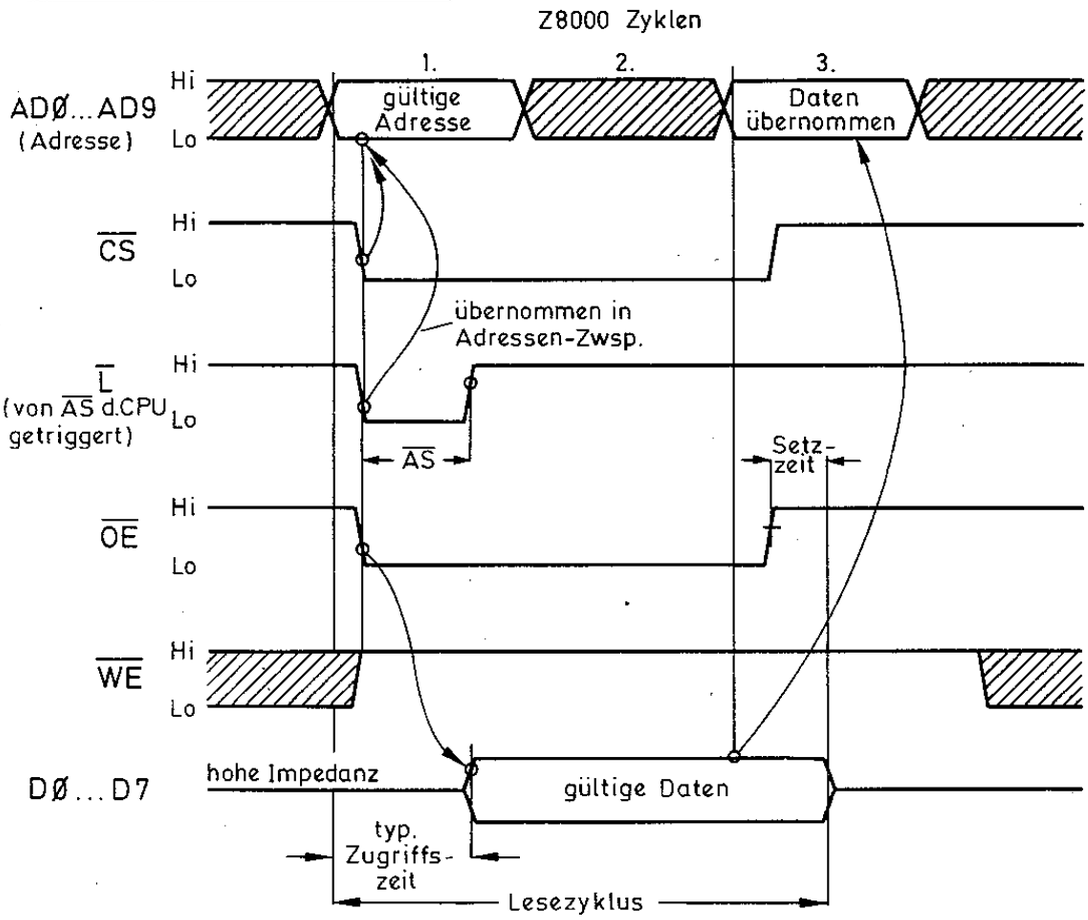

AD1...AD1Ø
AS
YØ (1K)
Y1 (2K)
Y2 (3K)
Y3 (4K)
Y4 (5K)
Y5 (6K)
Y6 (7K)
Y7 (8K)
R/W
DS
DØ... D7
bzw.
D8... D15
19
L
AØ
A9
4118
I/O1
I/O8
WE
OE
CS
21
20
18

Bild 4.9

Synchroner Lesezyklus 4118



Synchroner Schreibzyklus

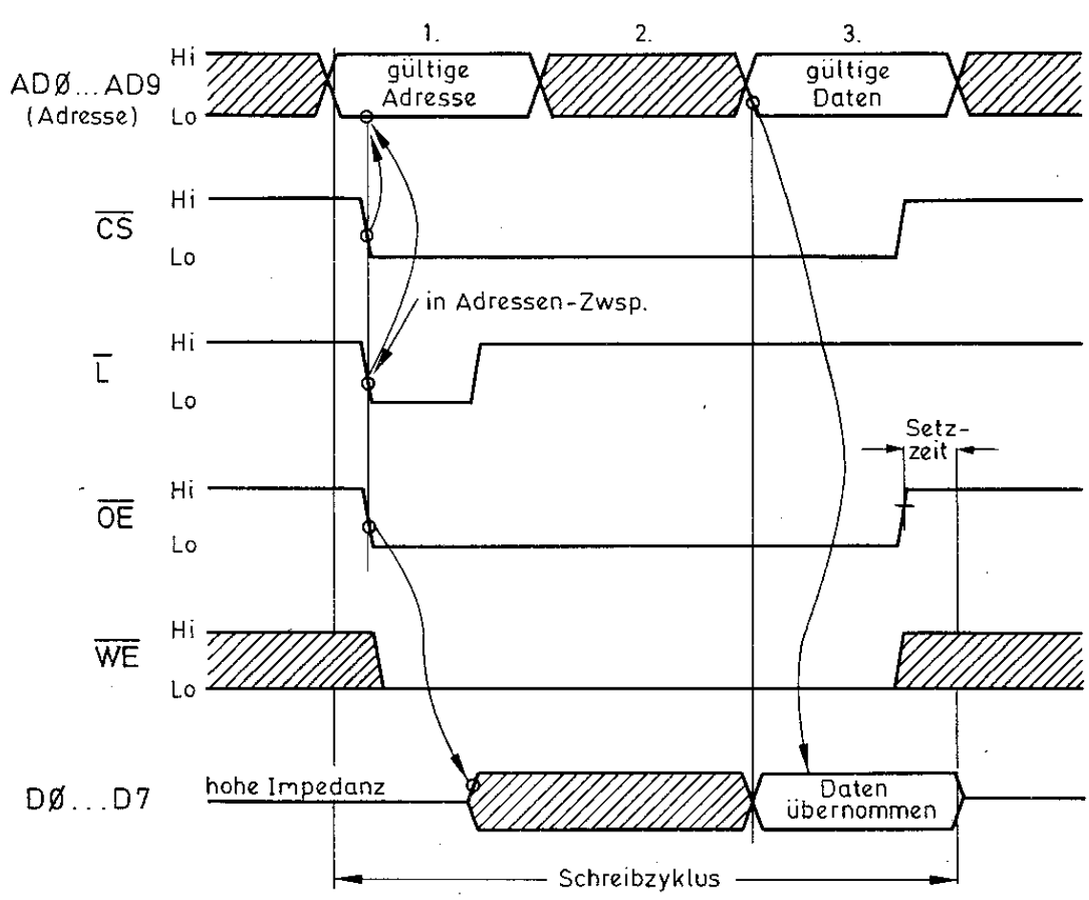


Bild 4.10

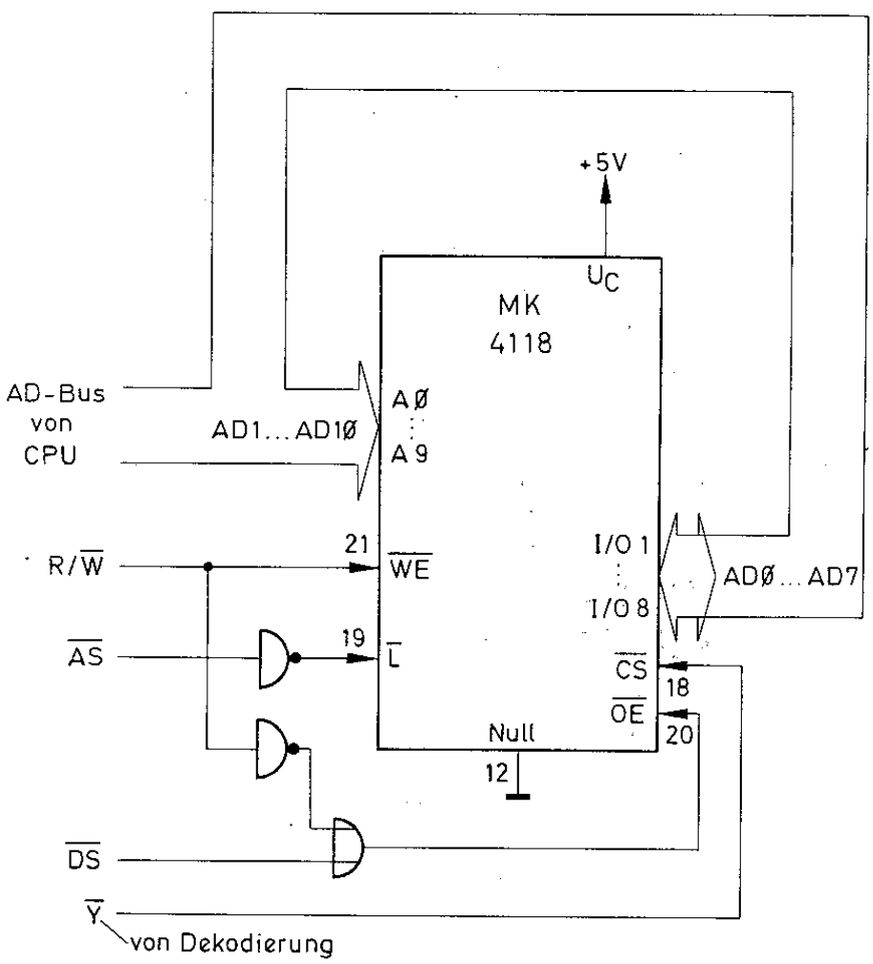

+5V
U_C
MK
4118
AD-Bus
von
CPU
AD1...AD1Ø
AØ
A9
R/W
21
WE
I/O 1
I/O 8
ADØ...AD7
AS
19
L
CS
18
OE
20
Null
12
DS
Y
von Dekodierung

Bild 4.11

wird die angelegte Adresse zwischengespeichert, solange $\overline{L}$ Lo bleibt. Gleichzeitig muß $\overline{CS}$ (und $\overline{OE}$ beim Lesen) aktiviert werden, damit der Baustein das adressierte Byte freigeben kann.

Man muß schon genau ins Detail gehen, um dieses RAM zu verstehen. Man unterscheidet bei diesem Speichertyp einen synchronen und asynchronen Lese- oder Schreibzyklus. Das Signaldiagramm in **Bild 4.10** zeigt einen synchronen Lese- und Schreibzyklus des 4118. Warum synchron und nicht asynchron gegenüber vielen anderen Speichertypen? Ein synchroner Speicherzugriff wird deshalb so genannt, weil in diesem Fall eine ausgegebene Adresse des Z 8000 zwischengespeichert wird. Die Adresse muß nicht mehr über die gesamte Dauer des Transfers am Speicherbaustein angelegt bleiben. Der von der CPU synchron ausgegebene Adreß-Strobe-Impuls $\overline{AS}$ bewirkt das Festhalten der Adresse, der, wie bereits gezeigt wurde, bei jeder Adreßausgabe ausgegeben wird. Die Zugriffszeit des 4118 wird wie bei jedem anderen Halbleiterspeicherbaustein vom Zeitpunkt $\overline{CS}$ aktiv (bei angelegter Adresse) gemessen. Der 4118 kann genauso auch asynchron betrieben werden. Dann muß $\overline{L}$ immer Hi sein. Somit muß auch die Adresse so lange angelegt bleiben, bis die Daten ausgegeben sind. In Bild 4.9 kann der synchrone zum asynchronen Zugriff einfach geändert werden, wenn $\overline{AS}$ statt von der CPU einfach auf Hi gelegt wird.

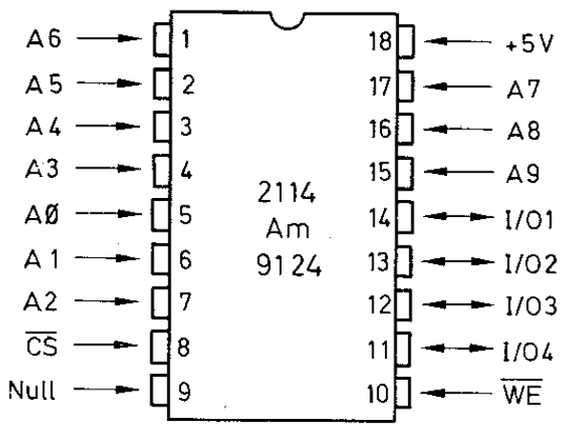
Wenn man die Eigenschaften des 4118 auf die Eigenwilligkeit des Z 8000 projeziert (gemeinsamer Adreß-/Datenbus), ist folgende Hardwarekonfiguration nach **Bild 4.11** möglich. Es ist nicht mehr nötig, für diesen Speichertyp den Bus zu trennen, weil die Übernahme und Ausgabe von der CPU synchron steuerbar ist. $\overline{AS}$ wird an $\overline{L}$ angeschlossen. Ist $\overline{AS}$ Lo, wird die ausgegebene Adresse festgehalten. Um hier ganz genau zu sein; die Adresse ist an der Lo ↑ Hi-Flanke von $\overline{AS}$ gültig. Das bedeutet, daß $\overline{AS}$ an $\overline{L}$ des 4118 invertiert herangeführt werden muß. Wenn man so will, ist das identisch zu Bild 4.3, in dem $\overline{AS}$ invertiert an die G-Eingänge der AmZ 8173 herangeführt wird. Die CPU schaltet auf Daten um, indem sie $\overline{DS}$ Lo setzt. $\overline{OE}$ wird nur bei einer Leseoperation aktiviert. Dies geschieht mit der ODER-Verknüpfung von R/$\overline{W}$ (invertiert mit $\overline{DS}$. Nur bei Daten lesen ($\overline{DS}$ gleich Lo, R/$\overline{W}$ gleich Hi) liegt Lo an $\overline{OE}$ an. In diesem Fall ist der AD-Bus der CPU auf Empfang geschaltet.

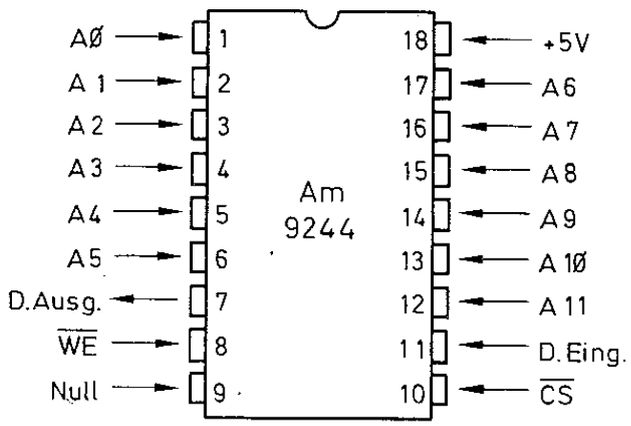
Daß der 4118 ein hochgradiger Speicherbaustein ist, zeigt sich nicht nur an dem eben Beschriebenen. Bereits die langsamste Ausführung mit der Bezeichnung MK 4118-4 ist mit 250 ns Zugriffszeit schnell genug, um einen Transfer mit einer 4 MHz-CPU-Version ohne zusätzlichen Wartezyklus der CPU zu garantieren. Für den superschnellen Z 8000A muß der Typ MK 4118-1/2 (120/150 ns Zugriffszeit) verwendet werden, um in gleicher Weise arbeiten zu können.

In **Bild 4.12** ist das Anschlußbild des jeweilig besprochenen Speichertyps dargestellt. Die Belegung des 4118 hat große Ähnlichkeit mit dem von 1 KB PROMs bzw. EPROMs.

4.2.3 Beispiel eines E/A-Speicherwerks mit dem Typ Am 9244 (4K x 1 Bit)

In Bild 4.12 ist das Anschlußbild des statischen Speicherbausteins Am 9244 dargestellt. Mit seinen 12 Adreßanschlüssen (entsprechend 4 K) paßt er nicht mehr in das Dekodierschema in Bild 4.7. Jedoch ist es kein Problem, die Adressen an der Dekodierlogik entsprechend zu ändern, damit die $\overline{Y}$-Leitungen in 4 K-Blöcken frei geben können. An die Binärdekoder LS 138 sind die Adreßleitungen A 13, A 14 und A 15 anzulegen. Darum geht es in diesem Abschnitt nicht. **Bild 4.13** zeigt eine nicht übliche Anordnung eines Speicherwerks. Es handelt sich um einen E/A-Speicher. Die CPU kann dieses 4 K Worte umfassende Spei-


A6
A5
A4
A3
AØ
A 1
A2
CS
Null
1
2
3
4
5
6
7
8
9
2114
Am
9124
18
17
16
15
14
13
12
11
10
+5V
A7
A8
A9
I/O1
I/O2
I/O3
I/O4
WE


AØ
A 1
A 2
A 3
A 4
A 5
D.Ausg.
WE
Null
1
2
3
4
5
6
7
8
9
Am
9244
18
17
16
15
14
13
12
11
10
+5V
A6
A 7
A 8
A 9
A 1Ø
A 11
D.Eing.
CS

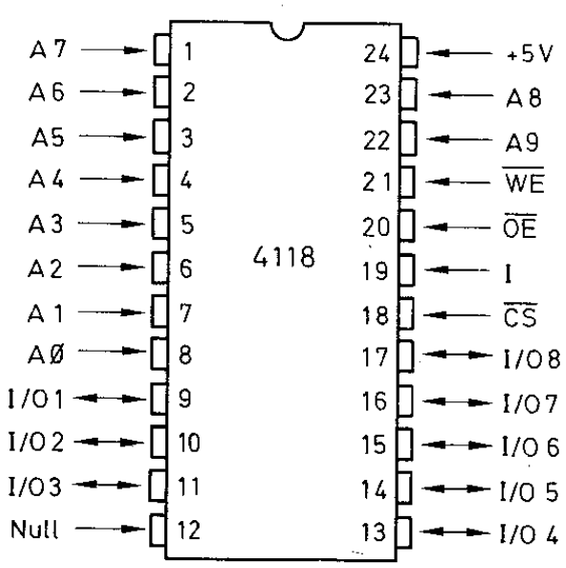

A 7
A 6
A 5
A 4
A 3
A 2
A 1
AØ
I/O 1
I/O 2
I/O 3
Null
1
2
3
4
5
6
7
8
9
10
11
12
4118
24
23
22
21
20
19
18
17
16
15
14
13
+5V
A 8
A 9
WE
OE
I
CS
I/O 8
I/O 7
I/O 6
I/O 5
I/O 4

Bild 4.12

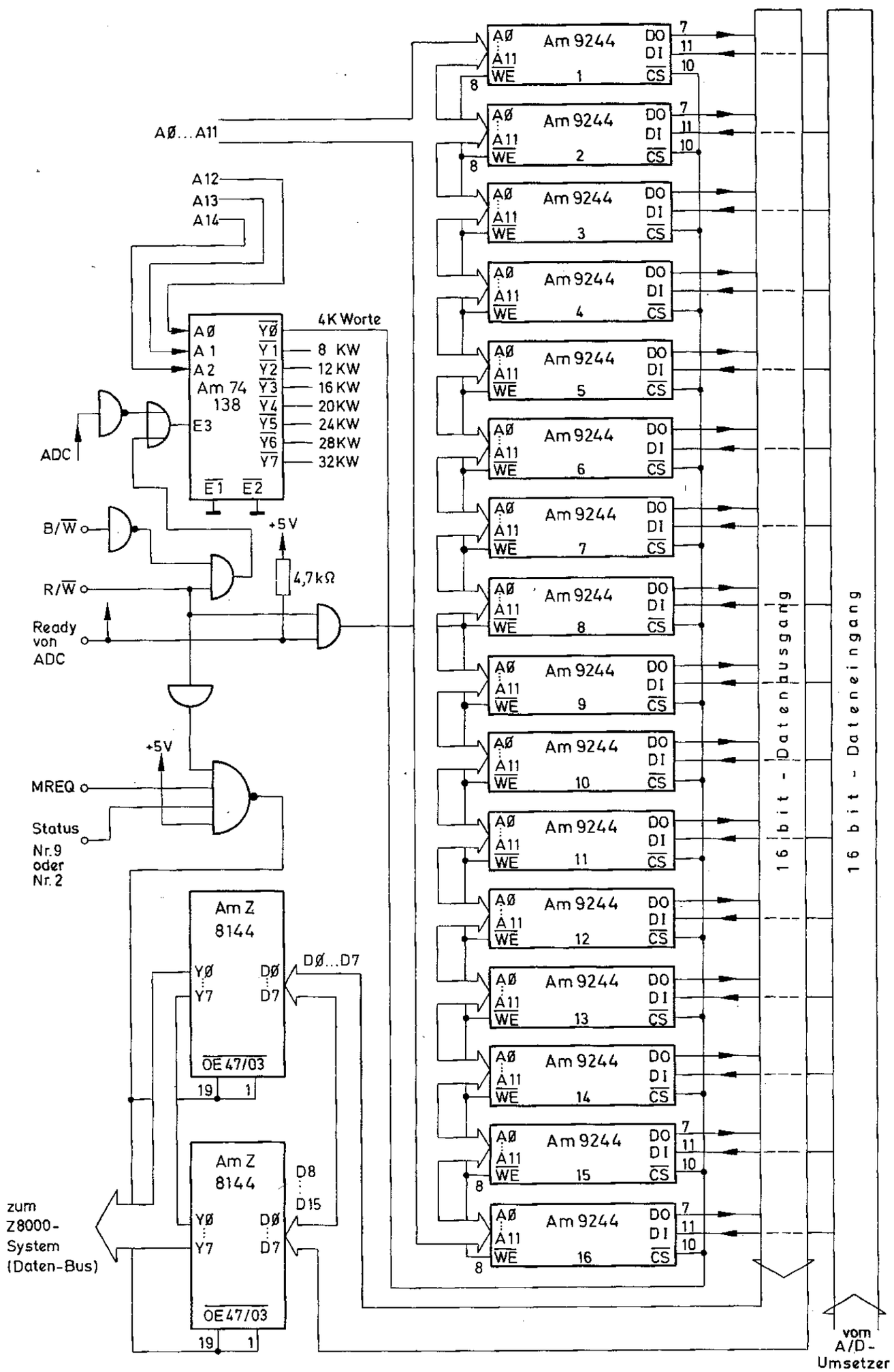

A0...A11
A12
A13
A14
Am 74 138
A0
A1
A2
E3
E1
E2
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
4K Worte
8 KW
12 KW
16 KW
20KW
24KW
28KW
32KW
ADC
B/W
R/W
Ready von ADC
+5V
4,7kΩ
MREQ
Status Nr.9 oder Nr.2
Am Z 8144
D0...D7
D8
D15
OE 47/03
19
1
zum Z8000-System (Daten-Bus)
Am 9244
A0
A11
WE
CS
DO
DI
7
11
10
8
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
16 bit - Datenbusgang
16 bit - Dateneingang
vom A/D-Umsetzer

Bild 4.13

cherwerk nur mit "Input"-Befehle lesen. Sie kann außerdem nur Worte auslesen und in keiner Form schreiben. Wozu das wird man fragen!

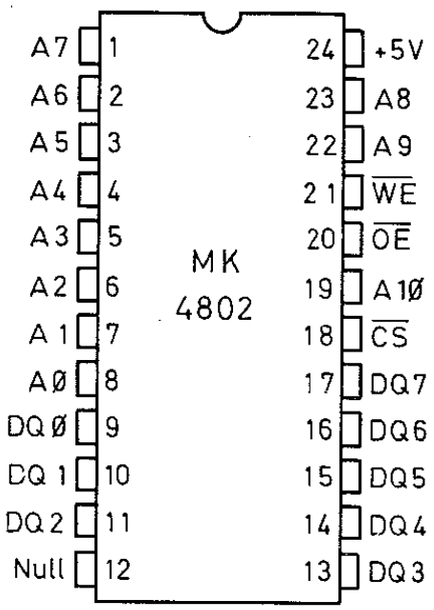
Dieser Speicher dient z.B. dazu, zwischen einer Analogdatenübernahme und der CPU als Pufferspeicher zu wirken. Ein Analog-Digital-Umsetzer lädt synchron (von der CPU unabhängig) gewandelte Werte in diesen Speicher . Die CPU holt diese Werte in ihren Arbeitsspeicher, wobei sie den Pufferspeicher nur mit ihren verschiedenen "Input"-Befehlen anspricht. Die Datenausgangstreiber AmZ 8144 sind nur dann zu öffnen, wenn $\overline{MREQ}$ gleich Hi, R/$\overline{W}$ gleich Hi und der invertierte Status Nr. 2 ausgegeben wird. Wie man weiß, ist das für Speicheroperationen nicht üblich. Ebenso wird die Steuerung für $\overline{WE}$ der Speicher zu R/$\overline{W}$ von der CPU mit einem "ready"-Signal des ADC logisch UND verknüpft. Ein Lo als Fertigmeldung des ADC kann zum Schreiben benützt werden.

Der 9244 RAM hat getrennte Datenein- und -ausgänge. Die Dateneingänge können beispielsweise von den digitalen Ausgängen des ADC angesteuert werden. Zur Adreßdekodierung ist nur ein Binärdekoder LS 138 notwendig, weil nicht zwischen gerader und ungerader Adresse unterschieden werden muß. In 4 K-Wortgruppen läßt sich daraus der E/A-Speicher im 64 K-Feld verschieben. Für die Adresse ist eine Multiplexschaltung notwendig, die zwischen einer CPU-Adresse und einer separaten Adreßlogik aus Zählern umschalten kann. Die Zähleradresse wird an die Speicherbausteine angelegt, wenn sie mit einem ADC-Wert programmiert werden sollen. Hierzu sind drei Bausteine 74LS157 (4 x 2 Bit-Datenselektoren) notwendig. Dieses zusätzliche Element ist in Bild 4.13 nicht mehr eingezeichnet, weil es in diesem Fall etwas vom Thema ablenkt. Es sei hier nur erwähnt, weil die Schaltung in Bild 4.13 nochmals in Kapitel 6 aufgegriffen wird.

4.2.4 Rein statischer Speichertyp MK 4802 (2K x 8 Bit)

Die Speicherdichte ist noch beträchtlich im Steigen begriffen. Diese Feststellung ist zwangsläufig zu treffen, wenn man sich genauer in den fachbezogenen Medien der Elektronik- und Computerbranche umsieht. Das vorliegende Buch ist sicher nicht der geeignete Rahmen, einen lückenlosen Überblick über alle Speichertypen zu offerieren, zumal das Angebot verschiedener Typen in diesem Kapitel ohnehin nicht kleinlich zu nennen ist. Gewissermaßen als Obergrenze für den Abschnitt der rein statischen Typen soll der MK 4802 von Mostek gelten, der in einem 24poligen Gehäuse immerhin 2 K x 8 Bit absolut sicher garantiert. Auch für diesen Baustein gilt die Basisdekodierung nach dem Schema in Bild 4.7. Ein Byteblock in der Größe 8 K benötigt vier Bausteine dieses Typs. An die 11 Adreßeingänge werden die CPU-Adressen A1 . . . A11 angelegt. An die Adreßdekoder LS 138 werden die Adreßleitungen A12 . . . A14 angelegt. Wenn man genau nach Bild 4.7 vorgehen will, kann man auf A14 verzichten und sie so verbinden, wie es dargestellt ist. Der Selekteingang A2 am LS 138 muß demnach auf Null gelegt werden. Somit sind nur noch die vier Chipselekt-Leitungen $\overline{Y0}$ . . . $\overline{Y3}$ von Bedeutung. Die schaltungstechnische Behandlung der $\overline{OE}$-Eingänge kann aus Bild 4.9 übernommen werden. 8 K Worte benötigen 8 Bausteine des Typs MK 4802.

Dieses RAM ist außerordentlich schnell. Der Typ MK 4802-90 als "langsamster" benötigt nur 90 ns und kann somit auch für den Z 8000A verwendet werden, ohne einen zusätzlichen Wartezyklus bei einem Zugriff einzufügen. **Bild 4.14** zeigt das komplette Anschlußbild dieses sehr schnellen, aber doch unkompliziert anwendbaren Speichertyps. Die technische Variante eines Busanschlusses nach Bild 4.11 ist jedoch mit dem 4802 nicht machbar, weil er keine Adressenzwischenspeicherung beinhaltet.


A7 1
A6 2
A5 3
A4 4
A3 5
A2 6
A1 7
A0 8
DQ0 9
DQ1 10
DQ2 11
Null 12
24 +5V
23 A8
22 A9
21 WE
20 OE
19 A10
18 CS
17 DQ7
16 DQ6
15 DQ5
14 DQ4
13 DQ3
MK
4802

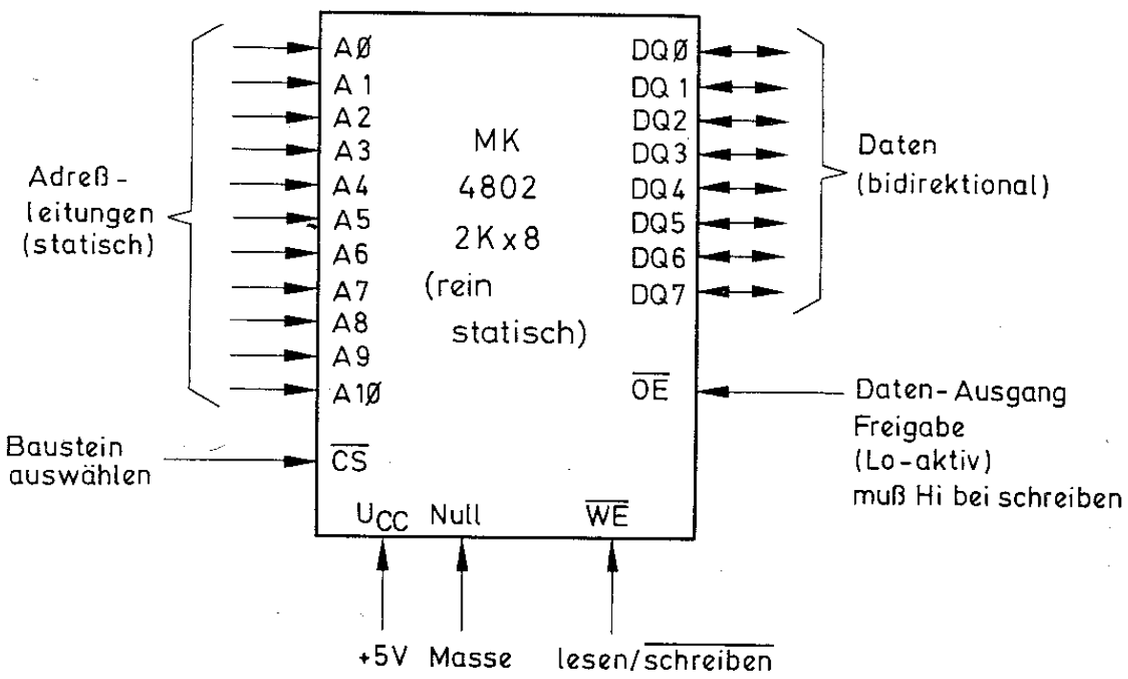

A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
CS
MK
4802
2K x 8
(rein
statisch)
DQ0
DQ1
DQ2
DQ3
DQ4
DQ5
DQ6
DQ7
OE
WE
UCC
Null
Adreß-
leitungen
(statisch)
Baustein
auswählen
Daten
(bidirektional)
Daten-Ausgang
Freigabe
(Lo-aktiv)
muß Hi bei schreiben
+5V
Masse
lesen/schreiben

Bild 4.14

4.3 ANSCHLUSS-SCHEMATIK FÜR QUASI-STATISCHE SCHREIB-/ LESESPEICHER

Die Hersteller von Halbleiterspeicher bemühen sich nicht nur um die Verdichtung von Speicherkapazität, sonder zusätzlich um leichte Handhabung und geringe Verlustleistung. Auch bei Computern ist man sehr auf geringen Leistungsverbrauch bedacht, nicht nur bei Autos. Mit trickreicher Technologie läßt sich Strom sparen. Die allein löst das Problem noch nicht. Bei Speicherbausteinen müssen eine Reihe von Faktoren zusammenwirken, um einen guten Kompromiß zu bilden. Sie sollen hohe Kapazität besitzen, wenig Strom verbrauchen, möglichst nur ein (übliche +5 V) Spannung erfordern, schnell sein, ohne viel zusätzliche Logik zu betreiben sein und zu guterletzt nichts kosten. Das klingt leicht nach "Perpetuum mobile". Die neue Generation von Speichern mit der Bezeichnung quasi statische RAMs verbindet jedoch die genannten Eigenschaften in ausgeprägt hohem Maß in einem Gehäuse, sodaß man sich wirklich viel davon versprechen darf.

4.3.1 Was sind quasi statische RAMs?

Speicherbausteine dieser Art sind nicht ganz statisch und nicht ganz dynamisch.

Wer sich in der Schaltungstechnik von RAMs (Random Access Memories) ein wenig auskennt, wird zu dieser Antwort sofort seine Bedenken anmelden, obgleich das Prädikat "quasi" darauf hindeuten mag. Ein RAM nach logischen Elementen ist entweder statisch auf der Basis von Flipflops (bistabile Kippstufe) oder dynamisch mit einer Kapazität als Speicherzelle. Halb Flipflop und halb Kapazität gibt es nicht. **Ein quasi statisches RAM ist rein dynamisch.** Der Speicherbaustein wirkt nur von außen statisch, weil das zyklische Aufladen der kapazitiven Speicherzellen von einer separaten Logik auf dem Chip selbst ausgeführt wird. Er unterscheidet zwischen zwei Arten eines Auffrischzyklus: Einzelrefresh vor oder nach einem Lese- bzw. Schreibzyklus und ein automatisch ablaufender interner Refresh mit 128 Zyklen in 2 ms. In dieser Zeit ist das gesamte dynamische Speicherfeld, bestehend aus Einzeltransistor mit einer Kapazität in der Drainleitung, neu aufgeladen. Die Steuerung des internen oder externen Auffrischzyklus erfolgt an einem Anschluß dieser Speicherbausteine. Die Organisation des externen Datenformats ist Byte orientiert. Zwei Bausteine bieten somit schon ein Minimum eines kompletten Arbeitsspeichers für den Z 8000. Die Bausteine verfügen über einen Generator zur Erzeugung einer negativen Vorspannung auf dem Substrat und erlauben demzufolge einen Betrieb mit einer einzigen Versorgungsspannung (+5 V ± 10%). Es wurde bereits genannt, daß dynamische Speicherzellen sehr wenig Strom verbrauchen.

Quasi statische RAMs verbrauchen durchweg so wenig Strom, daß sie längere Zeit mit einer Taschenlampenbatterie zu betreiben sind. Es wird überdies zwischen aktivem und inaktivem Zustand unterschieden, das sich in einer weiteren Reduzierung der Belastung der Stromquelle im inaktiven Zustand etwa um den Faktor 6 bis 7 äußert. Der Betrieb der quasi statischen RAMs erfordert kei-

ne Adreßmultiplexer, wie sie für typische dynamische RAMs benötigt werden.

Folgende Typen quasi statischer RAMs sind bereits verfügbar:

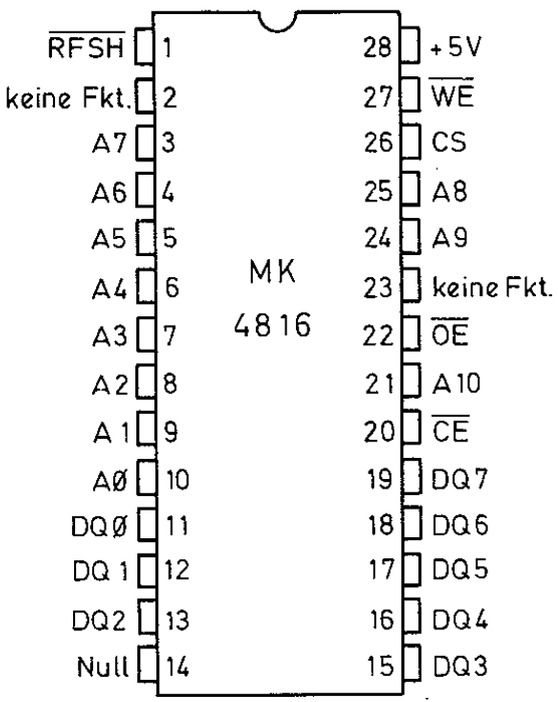
1K x 8 Bit	MK 4808, MK 4809
2K x 8 Bit	MK 4816
4K x 8 Bit	Z 6132

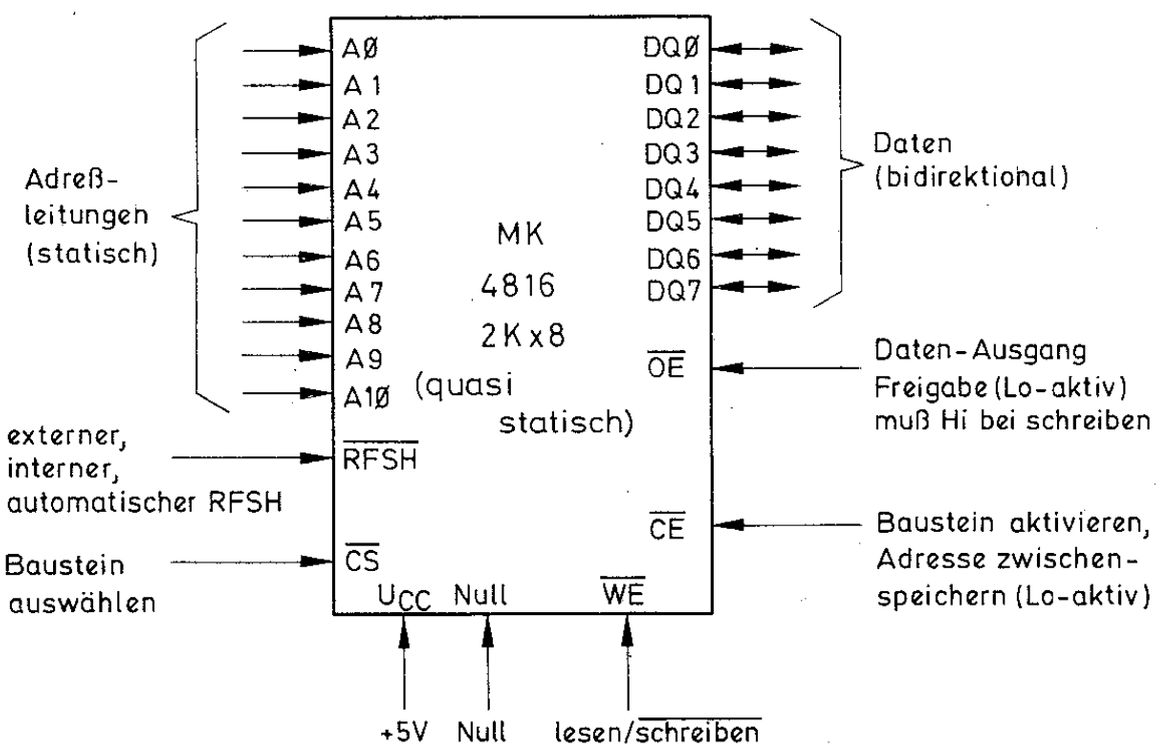
An einem Speichertyp des Formats 8K x 8 Bit wird noch experiementiert. Er soll jedoch 1981 auf den Markt kommen. Quasi statische RAMs sind für Hardwarekonfigurationen des Busses in der Art nach Bild 4.11 vorbereitet.

4.3.2 Beispiele für den Busanschluß mit den Typen MK 4816 und Z 6132

Der 2 K Byte quasi statische RAM MK 4816 von Mostek wird in Zugriffszeiten von 150 . . . 300 ns angeboten. 4 Bausteine ergeben einen 8 K Byteblock. Ein Speicherwerk mit 4816 RAMs ist nach dem Schema in Bild 4.7 aufzubauen, wenn die Adressenzuführung geändert wird, wie sie für den 2 KB-Typ 4802 beschrieben wurde. Für den Einsatz des 4816 sind in der Schaltung nach Bild 4.7 noch weitere Änderungen vorzunehmen. **Bild 4.15** zeigt das komplette Anschlußbild des MK 4816. Das Chipselekt-Signal CS ist für diesen Speicher H-aktiv und muß Hi sein, wenn an dem Adreßspeichereingang $\overline{CE}$ während einer Hi ↓ Lo-Flanke die angelegte Adresse zwischengespeichert werden soll. Nach einer Haltezeit von 20 . . . 35 ns und des gleichen Zeitraums für das CS-Signal liegt die Adresse intern fest. $\overline{CE}$ ist ein Adreßtakteingang, mit dem unbedingt gearbeitet werden muß. Er speichert nicht nur bei seiner Aktivierung eine Adresse, sondern schaltet auch intern zwischen Zeilen- und Spaltenadresse um. **Bild 4.16** zeigt einen Lesezyklus mit dem 4816 RAM. Der Zyklus ist auf die Signalausgabe des Z 8000 zugeschnitten. Es gibt zwei Möglichkeiten, $\overline{CE}$ für eine Adreßübernahme auf Lo zu ziehen. Die erste, sicherlich kritische Methode ist, von der Hi ↓ Lo-Flanke des $\overline{AS}$-Signals der CPU eine feste Verzögerung einzubauen, die etwa nach 120 ns $\overline{CE}$ auf Lo zieht. Man könnte damit $\overline{CE}$ gewissermaßen vorausschauend aktivieren und die Zugriffszeit für den Speicher um einige Nanosekunden ausdehnen. Diese Methode will jedoch überlegt sein. Die sichere Methode ist in jedem Fall, die Lo ↑ Hi-Flanke von $\overline{AS}$ als gültigen Adreßzeitpunkt zu nehmen, CS zu aktivieren und mit einem Inverter am Selektausgang des Dekoders $\overline{CE}$ anbinden. Für den 4816-2 muß CS noch 20 ns aktiv bleiben, damit die Adresse sicher übernommen wird. Die Zugriffszeit auf den Speicher beginnt mit der Hi ↓ Lo-Flanke von $\overline{CE}$. Beim Lesen muß zur Ausgabe der Daten $\overline{OE}$ aktiviert werden. Man sollte das mit $\overline{DS}$ der CPU tun. Da nun die besagte Bedingung besteht, $\overline{OE}$ nur bei R/$\overline{W}$ gleich Hi zu aktivieren, muß $\overline{DS}$ und R/$\overline{W}$ auf $\overline{OE}$ so verknüpft werden, wie es in Bild 4.9 oder 4.11 dargestellt ist. Für den Typ MK 4816-2 erhält man an seinen DQ-Anschlüssen nach 150 ns den Inhalt auf die angelegte Adresse.

Bild 4.17 zeigt einen Schreibzyklus mit dem 4816 RAM. Die CPU-Signale R/$\overline{W}$ und $\overline{DS}$ sind ODER zu verknüpfen. R/$\overline{W}$ muß vorher noch invertiert werden. Das gemeinsame Signal steuert $\overline{OE}$. Beim Schreiben muß $\overline{OE}$ immer inaktiv sein, weil sonst Ausgang auf Ausgang trifft. Die Zeit von 20 ns nach der Hi ↓ Lo-Flanke von $\overline{CE}$ muß eingehalten werden. Diese Zeit ist als Haltezeit anzusehen.





extern ($\overline{RFSH}$ = Hi): Zeilenrefresh AØ ... A6 alle 2 ms;
intern durch Lo-Puls (35 ... 65 ns);
automatisch ($\overline{RFSH}$ = Lo), Refr.-Zyklus alle 15 µs.

Bild 4.15

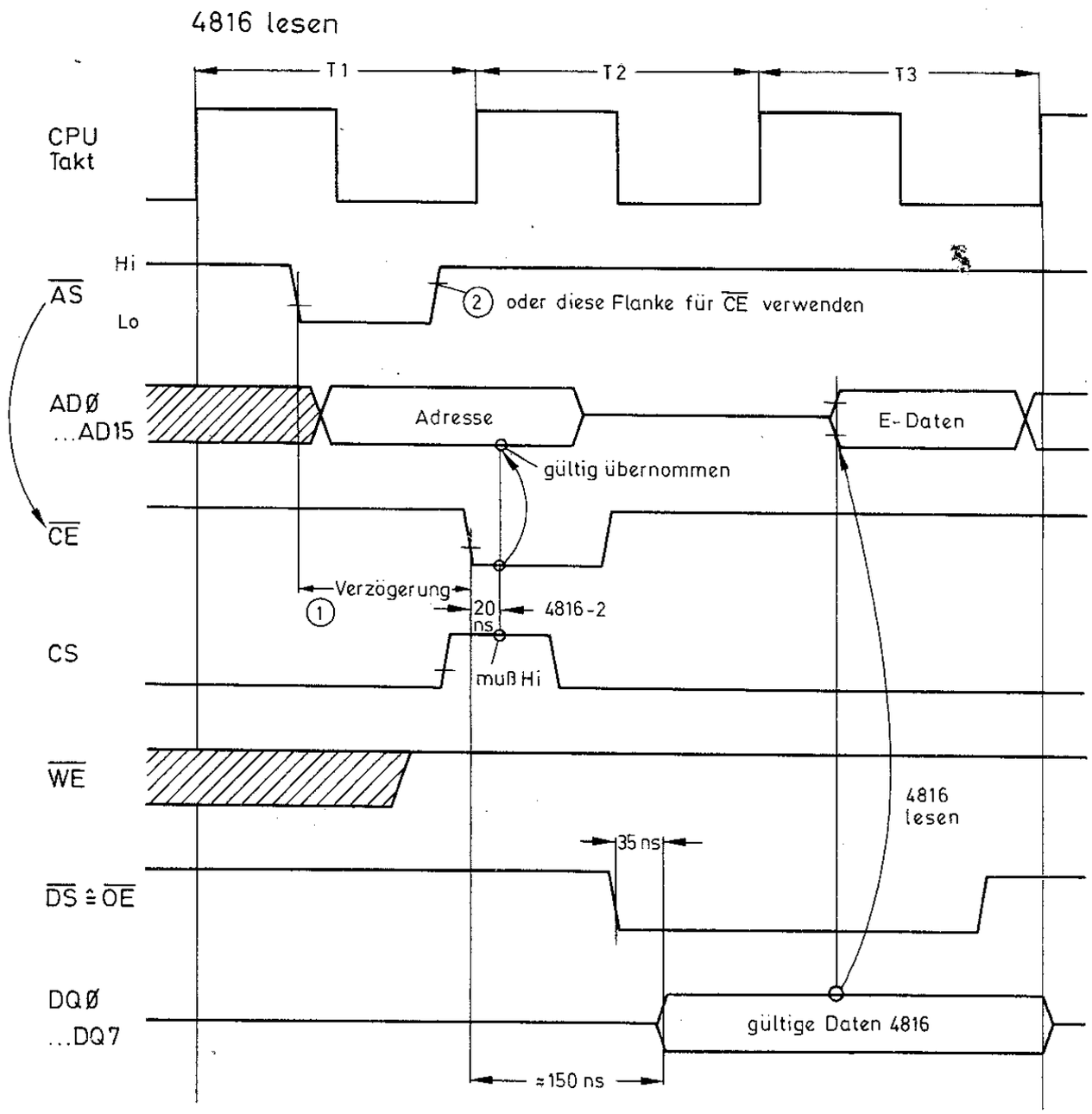

4816 lesen
T1
T2
T3
CPU Takt
Hi
Lo
AS
2 oder diese Flanke für CE verwenden
AD0 ...AD15
Adresse
E-Daten
gültig übernommen
CE
Verzögerung
1
20 ns
4816-2
CS
muß Hi
WE
4816 lesen
35 ns
DS ≙ OE
DQ0 ...DQ7
gültige Daten 4816
≈150 ns

Bild 4.16

4816 schreiben

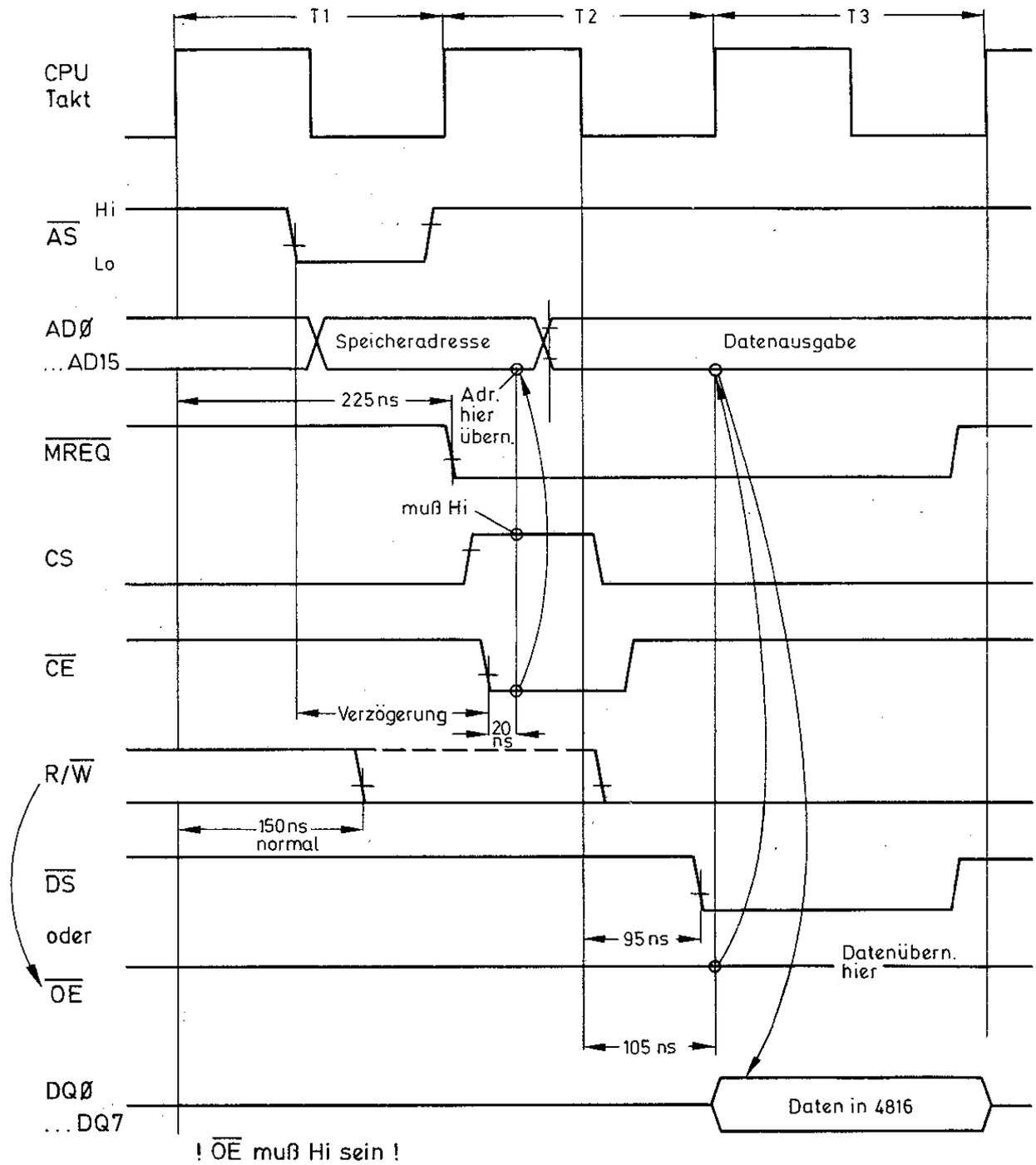


Bild 4.17

In **Bild 4.18** ist die Schaltung eines Speicherwerks dargestellt, das mit zwei MK 4816-2 eine Kapazität von 2 K Worte ergibt. Die Schaltung weicht prinzipiell etwas von der in Bild 4.7 ab, weil hier mit einem gemeinsamen Adreß-/Datenbus gearbeitet wird. Die Dekodierung ist für gerade und ungerade Adreßworte vorhanden. Im gegebenen Fall ist es nicht mehr möglich, mit einfachen LS 138-Dekodern zu arbeiten; es sind zu diesem Zweck zwei Bausteine des Typs AmZ 8136 eingesetzt. Sie bieten eine Speichermöglichkeit der Auswahladreßwerte an den A-, B-, C-Eingängen. Drei Adressen werden zur Lo ↑ Hi-Flanke von $\overline{AS}$ an CP gespeichert. Die Taktfreigabe $\overline{CS}$ ist auf Lo zu legen. An dem POL-Eingang lassen sich die Selektausgänge auf Lo- oder Hi-aktiv festlegen. Da man für CS der Speicherbausteine Hi-aktiv benötigt, muß POL statisch Hi sein. **Bild 4.19** zeigt das Anschlußbild für den Adreßkodierer 8136. An $\overline{CE}$ muß immer dann Lo liegen, wenn Selektionsadressen an einer Lo ↑ Hi-Flanke von CP gehalten werden sollen. Die Selektionsausgänge sind über Lo an $\overline{OE}$ freizugeben.

Die erste Bedingung an $\overline{G1}$ für eine Selektion der RAM-Bausteine ergibt sich aus $\overline{MREQ}$ gleich Lo. Die zweite Bedingung wird durch B/$\overline{W}$ festgelegt. Ist B/$\overline{W}$ gleich Hi, kann Y0 aktiviert werden. Dies ist dann nur noch von einer geraden oder ungeraden Adresse abhängig. Y0 gelangt direkt an CS und löst die höchste Priorität des Chipselekt. Damit ist ein Baustein aktiviert und benötigt statt 5 mA jetzt 30 mA Strom. Unmittelbar dazu ist der Eingang $\overline{CE}$ des RAM zu aktivieren. Eine Verzögerung von etwa 120 ns nach der Hi ↓ Lo-Flanke von $\overline{AS}$ kann dies bewirken. Die zu diesem Zeitpunkt sicher gültige Adresse wird gespeichert. Der nächste Vorgang ist die Behandlung der Daten, die in oder aus dem Speicher fließen. Aus dem jeweiligen Status muß $\overline{OE}$ abgeleitet werden. R/$\overline{W}$ ist der invertierte Wert von R/$\overline{W}$, der mit $\overline{DS}$ logisch ODER verknüpft wird.

An der gezeigten Schaltung sind zwei Dinge auszusetzen. Das Speicherwerk kann in dieser Form nur im Byte-Modus aktiviert werden. B/$\overline{W}$ muß immer Hi sein, um generell eine Freigabe zu erlangen. Die zusätzlichen Gatter vor den G2-Eingängen der 8136 beheben das. Bei B/$\overline{W}$ gleich Lo (Wort-Modus) werden grundsätzlich beide G2-Bedingungseingänge freigegeben. Bei B/$\overline{W}$ gleich Hi (Byte-Modus) wird nur ein Dekoder (gerade, ungerade) freigegeben. **Bild 4.20** zeigt die entsprechenden Schaltungsänderungen. B/$\overline{W}$ ist zu invertieren und auf zwei ODER-Gatter zu legen. Ein weiterer Punkt ist die schon genannte Verzögerung von $\overline{AS}$ an $\overline{CE}$. Geht man nach Bild 4.16 von der Möglichkeit (2) aus, indem man die Lo ↑ Hi-Flanke von $\overline{AS}$ verwendet, läßt sich daraus auch $\overline{CE}$ erzeugen. An den Y-Ausgängen der Dekoder sind jeweils Inverter zu schalten, deren Ausgänge an $\overline{CE}$ des gleichen Speicherbausteins anzuschließen sind. Diese Schaltungsmaßnahme ist für jeden Y-Ausgang zu treffen. **Bild 4.21** zeigt die Umgehung der Verzögerung für die Schaltung in Bild 4.18. Jeweils an Y0 wird $\overline{CE}$ erzeugt.

Das Auffrischen des 4816 erfordert, wie genannt, 128 Zyklen in 2 ms. Jeder aktive Schreib- oder Lesezyklus zählt zugleich als Auffrischzyklus. Die verbleibenden Zyklen müssen mit einer Auffrischadresse durchgeführt werden, um 128 zu erreichen. Entweder liefert die CPU diese Auffrischadresse bzw. den Zyklus oder der 4816 liefert selbst die Adresse. Man kann die Betriebsart am $\overline{RFSH}$-Anschluß wählen. Wird er statisch auf Hi gelegt, sind zu den aktiven Zyklen entsprechend Zeilenadreßwerte (A0 . . . A6) anzulegen, die insgesamt 128 in 2 ms nicht überschreiten müssen. Dieser externe Auffrisch-Modus ist mit der Z 8000-CPU leicht durchführbar, wie bereits in Kapitel 2 beschrieben wurde. Der interne Refresh ist mit $\overline{RFSH}$ pulsierend durchzuführen. Diese Methode ist nur bei quasi statischen RAMs machbar. Ein pulsierendes Lo-Signal gibt bei jedem Lo eine interne Auffrischadresse frei, die von einem Zähler erzeugt

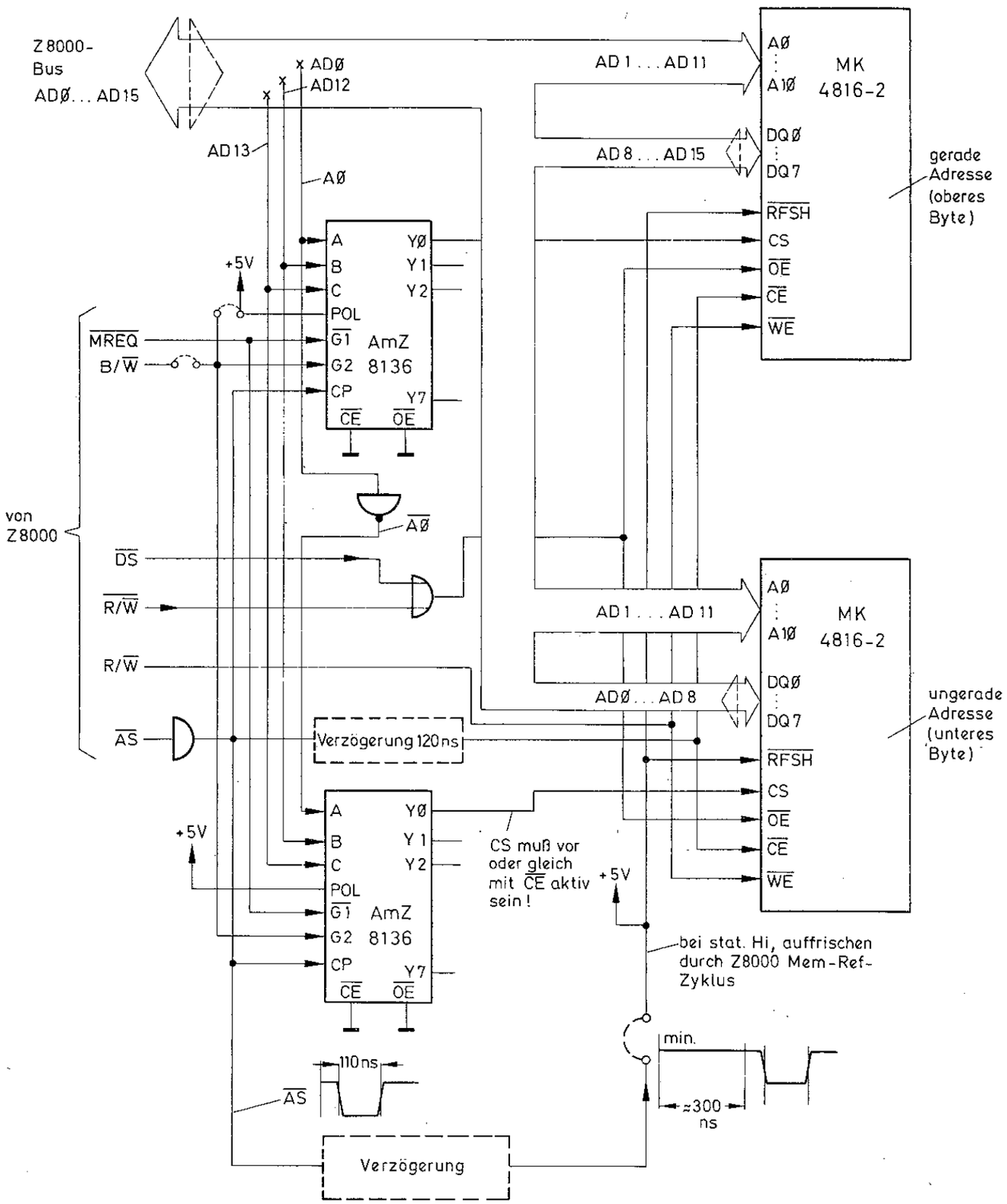

Z 8000-
Bus
AD Ø... AD 15
AD Ø
AD 12
AD 13
A Ø
AD 1 ... AD 11
AD 8 ... AD 15
A Ø
A 10
MK
4816-2
DQ Ø
DQ 7
RFSH
CS
OE
CE
WE
gerade
Adresse
(oberes
Byte)
+5V
A
B
C
POL
G1
G2
CP
CE
OE
Y Ø
Y 1
Y 2
Y 7
AmZ
8136
MREQ
B/W
von
Z 8000
A Ø
DS
R/W
R/W
AS
Verzögerung 120 ns
AD Ø ... AD 8
ungerade
Adresse
(unteres
Byte)
CS muß vor
oder gleich
mit CE aktiv
sein !
bei stat. Hi, auffrischen
durch Z8000 Mem-Ref-
Zyklus
min.
110 ns
≈300
ns
Verzögerung

Bild 4.18

Seitenadreßdekodierung mit
Register - 8 bit - Dekodierer
(AD-Bus kompatibel)

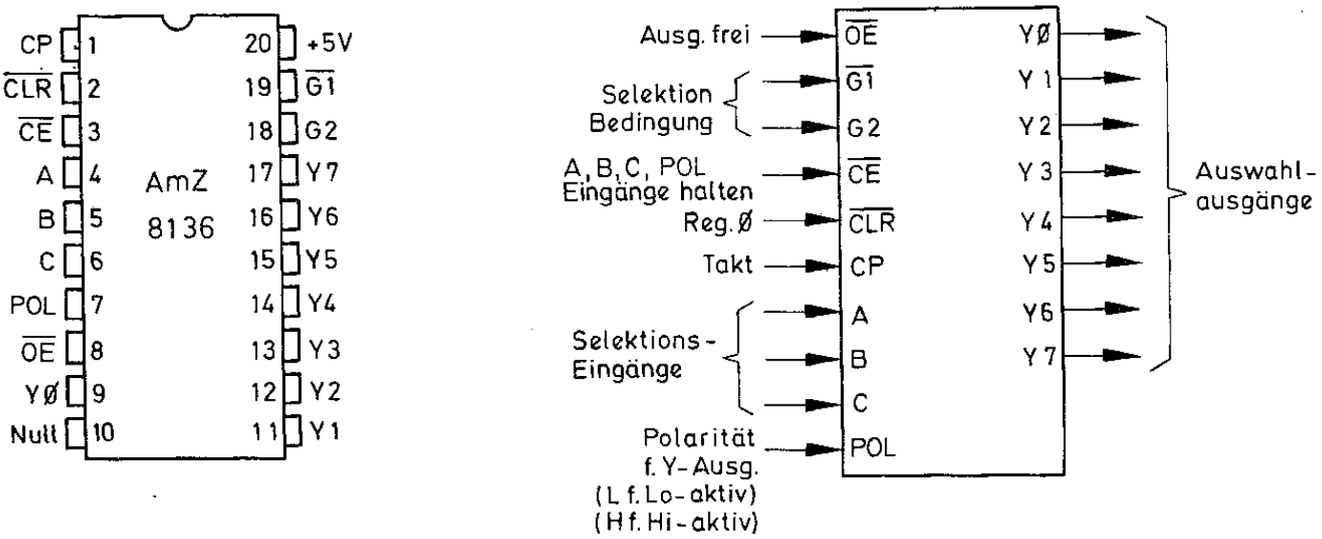


Bild 4.19

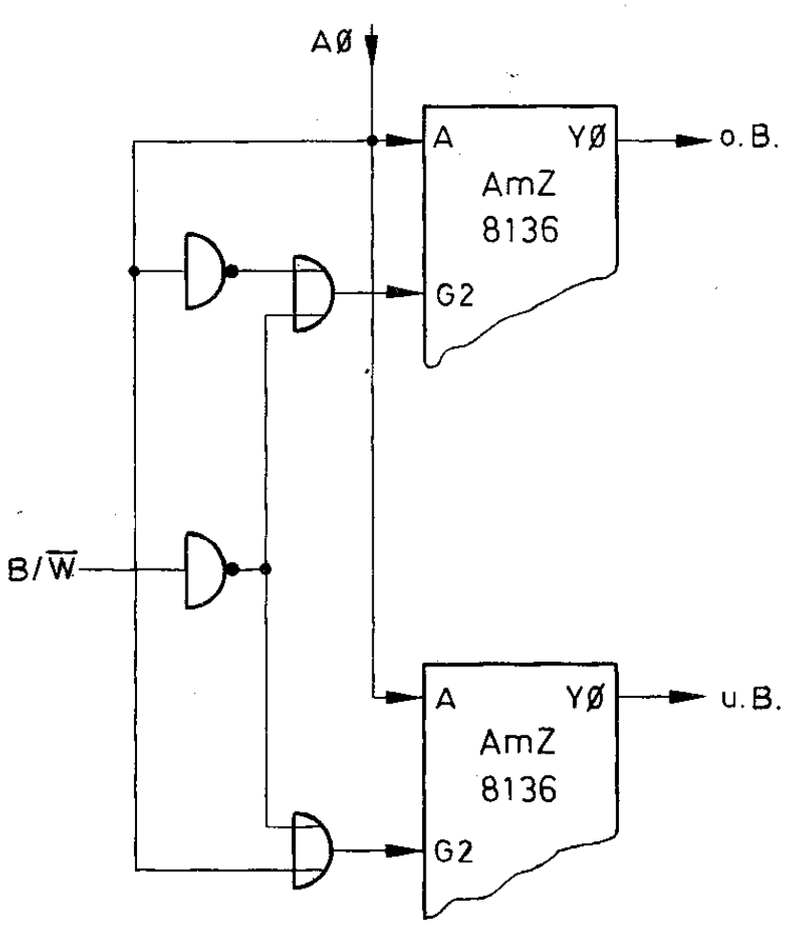


Bild 4.20

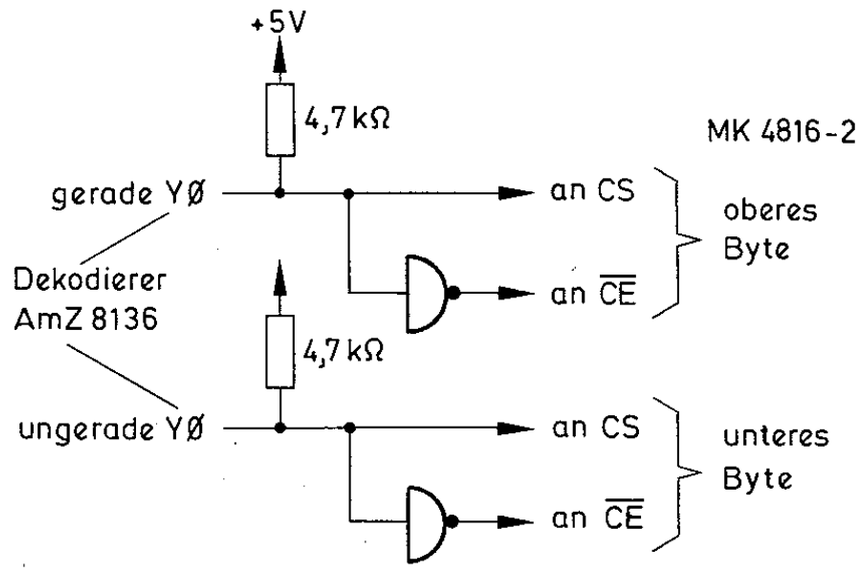


Bild 4.21

wird. Dazu würde man selbstverständlich das Adreßstrobe-Signal $\overline{AS}$ verwenden. Kommt $\overline{AS}$ von der CPU, ist es unerheblich, ob die Adresse auch noch von ihr oder von dem Speicherbaustein kommt. Es ist deahlb zu empfehlen, $\overline{RFSH}$ beider Bausteine auf Hi zu legen und die Refreshadresse von der CPU zu verwenden. Man erspart sich auf diese Weise eine weitere Verzögerungsschaltung.

Zu dem Netzwerk in Bild 4.18 sei noch bemerkt, daß die Verdrahtung von A0 an die Selekteingänge nicht empfehlenswert ist, weil die Stetigkeit der Selektion an den Y-Ausgängen nicht mehr gegeben ist. Man sollte nach dem Vorbild in Bild 4.7 mit hohen Adreßleitungen A . . . C belegen. Wenn man nur einen Selektionsausgang benötigt, ist das unerheblich.

Für das Jahr 1980 ist der quasi statische 4K x 8 Bit-Speicherbaustein Z 6132 als der mit der größten Speicherkapazität anzusehen. **Bild 4.22** zeigt das komplette Anschlußschema. Der Baustein arbeitet mit einer Spannung (+5 V ± 10%) und enthält auf dem Chip ebenso einen Bias-Generator, der auf dem Substrat eine negative Spannung von −3 V erzeugt. Die Anschlußbezeichnung des 6132 ist mit dem 4816 sehr ähnlich. Die Signale $\overline{BUSY}$, AC, $\overline{DS}$ und $\overline{CS}$ sind jedoch mit $\overline{RFSH}$, CS, $\overline{OE}$ und $\overline{CE}$ des 4816 nicht identisch. Man muß zumindest bemerken, nicht voll identisch. Der Ausgang $\overline{BUSY}$ an Pin 1 des 6132 ist im Vergleich zu Pin 1 des 4816 ein Eingang. $\overline{BUSY}$ ist ein offener Drain-Ausgang, an dem der Auffrisch-Modus festzulegen ist. Der 6132 kann in zwei Auffrischarten betrieben werden. Die einfachste Methode wird vorgegeben, wenn $\overline{BUSY}$ statisch auf Lo gelegt wird. Ein Speicherzyklus (schreiben, lesen) zieht in diesem Fall einen Auffrischzyklus hinterher. Ein kompletter Zyklus besteht aus

- Speicher schreiben oder lesen,
- Vorladen der Speicherzellen,
- Auffrischen durch den internen 7-bit-Adreßzähler; jeweils eine der 64 Zeilen eines Blocks,
- Vorladen der Speicherzellen.

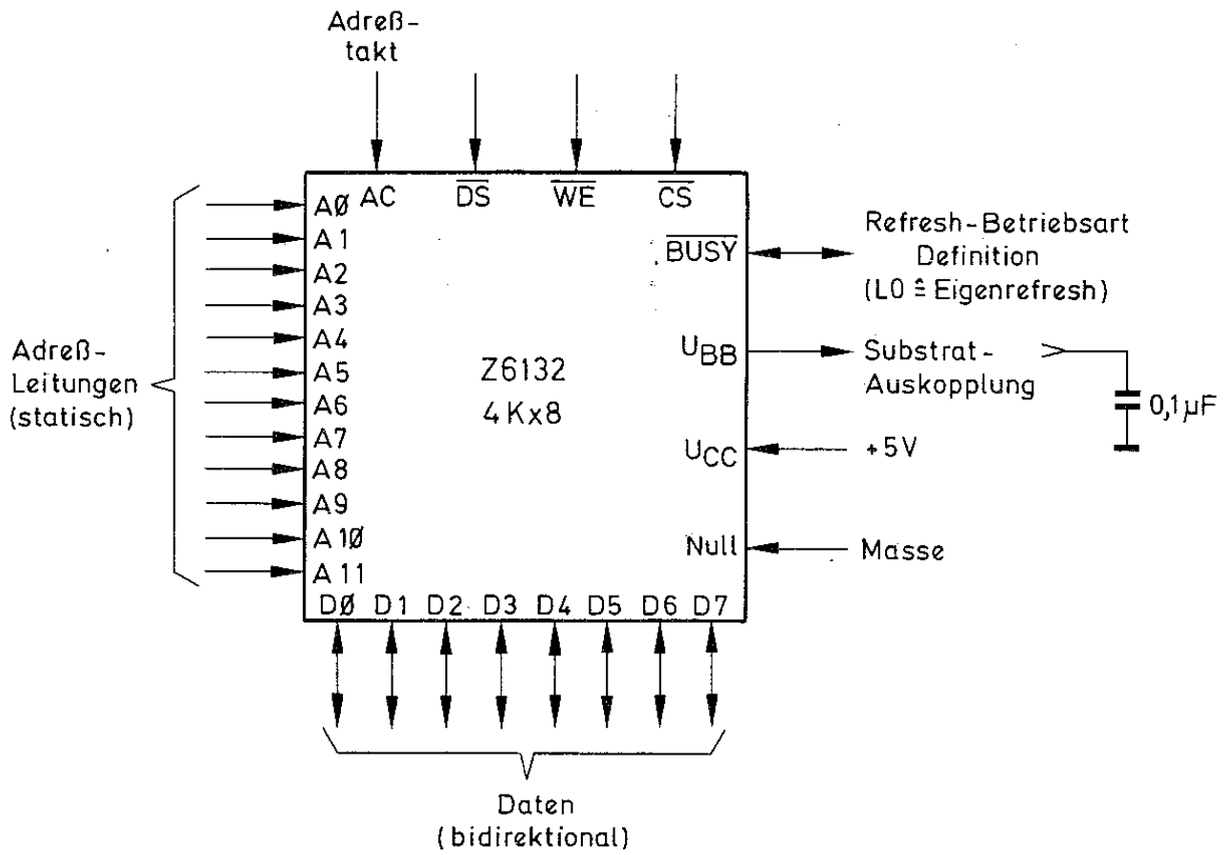
Dieser Langzyklus läuft automatisch ab, ohne daß äußere Hardware unterstützend einwirken muß.
Wird $\overline{BUSY}$ über einen Widerstand auf Hi gezogen, wird ebenfalls vom internen Adreßzähler eine Refreshadresse ausgegeben. Beide Speicherblöcke auf dem Chip (64 Zeilen, 128 Spalten) werden bei nicht angewähltem Baustein ($\overline{CS}$ gleich Hi) zyklisch aufgefrischt. Nach jedem Zyklus wird der Auffrisch-Adreßzähler inkrementiert. Ist der Baustein aktiviert ($\overline{CS}$ gleich Lo bei einer Lo ↑ Hi-Flanke am Eingang AC), wird auf Grund

der Adreßleitung A0 ausgewählt, welche der beiden Blöcke adressiert ist. Der nicht angewählte Block wird vom Adreßzähler aufgefrischt, der anschließend inkrementiert wird.

Der Eigenrefresh des 6132 zeigt noch eine weitere, echte Raffinesse auf. Ist der Baustein über längere Zeit hin aktiviert, z.B. bei Suchoperationen, Tabellenmanipulation . . ., wird in einem Zykluszähler mitgezählt, wann der nächste Refresh erfolgen soll. Ist der Zeitpunkt dafür erreicht, setzt die Refresh-Anforderungslogik, in der der Zykluszähler integriert ist, den $\overline{BUSY}$-Ausgang auf Lo. Ist $\overline{BUSY}$ mit $\overline{WAIT}$ der CPU verbunden, kann sie dadurch kurz angehalten werden, damit intern ein Refreshzyklus (Langzyklus-Refresh mit den genannten 4 Phasen, etwa 500 ns Dauer) ausgeführt werden kann. Der Speicherzykluszähler zählt bis auf 18 Schreib- oder Lesezyklen, nach denen der Refreshzähler eine Auffrischadresse ausgibt. Wird der Auffrischzähler nach der Adreßausgabe inkrementiert, wird zugleich der Zyklenzähler auf Null gesetzt und der Vorgang beginnt von neuem.

In diesem Punkt ist der Z 6132 dem MK 4816 wirklich überlegen. Außerdem erlaubt dieses Zilog-Produkt das CPU-Refreshzählerregister außer Betrieb zu setzen. Beim 4816 ist das nicht möglich, weil die Verwendung seines Refreshzählers schwieriger zu steuern geht. Ein WAIT für die CPU während eines Selbstrefresh muß extern erzeugt werden. Der Z 6132 kann nach außen hin wirklich rein statisch erscheinen, obwohl er intern voll dynamisch aufgebaut ist.

Bild 4.23 zeigt die Schreib- und Leseoperation des 6132. Der Eingang AC hat die Eigenschaft eines Adreßtakts. AC kann direkt mit $\overline{AS}$ der CPU verbunden werden, weil an AC eine Lo↑Hi-Flanke für die Übernahme einer Adresse sorgt, wenn $\overline{CS}$ Null war. Während dieser Flanke spielt sich einiges ab. Wird AC wieder Hi, muß beim Lesen $\overline{WE}$ nur über die kurze Haltezeit von etwa 80 ns Hi sein. Der Eingang $\overline{DS}$ muß während der Adreßübernahme ebenfalls Hi sein. Die Datenausgabe erfolgt, nachdem $\overline{DS}$ gleich Lo geworden ist. Die Zugriffszeit bestimmt sich aus der Lo↑Hi-Flanke von AC bzw. $\overline{AS}$ bis zum Ausgangssignal an D0 . . . D7. Betrachtet man den Signalverlauf von $\overline{AC}$ und $\overline{DS}$, ist unschwer erkennbar, daß diese Steuereingänge des Speichers direkt mit $\overline{AS}$ und $\overline{DS}$ der CPU zu verbinden sind. Für Interfaces mit diesem Speicherbaustein ist das eine absolute Erleichterung. Man erkennt jetzt auch den Unterschied zwischen $\overline{OE}$ des 4816 und $\overline{DS}$ des 6132. $\overline{OE}$ benötigt eine Verknüpfung von $\overline{DS}$ mit R/$\overline{W}$ der CPU, was bei $\overline{DS}$ am 6132 nicht notwendig ist. Ferner hat $\overline{CS}$ ebenso Priorität, wie bei allen bisherigen Speicherbausteinen. Hingegen ist bei CS vom 4816 (Pin 26) die höchste Priorität, nicht $\overline{CE}$ (Pin 20) Würde CS ständig Hi gelegt, wäre der 4816 ständig aktiviert. Insgesamt muß man feststellen, daß der Z 6132 sehr unkompliziert zu behandeln ist. Dies läßt sich auch leicht denken, daß die Interfacebehandlung auf den Z 8000 zugeschnitten ist, wenn "alles aus der gleichen Quelle kommt". Für den 6132 kann bei einer Schreiboperation noch ein besonderer Modus eingeführt werden, das Schreiben mit Inhibit. Das bedeutet, daß eine im Ansatz befindliche Schreiboperation noch kurz zuvor vereitelt werden kann, indem $\overline{WE}$ vor der fallenden Flanke von $\overline{DS}$ wieder auf Hi gesetzt wird. Warum das, wird man fragen. Hier gibt es eine klare Antwort. Es hängt mit dem Speicherorganisationsbaustein Z 8010 MMU zusammen. Durch die außerordentlich präzise Segmentdefinition der MMU ist es möglich, bei einem willkürlichen Segment einen Schreibschutz vorzusehen. Die MMU erzeugt ein spezifisches Signal mit der Bezeichnung "Suppress" (SUP), das nach eingehender Prüfung eines Segmentbeschreibungsregisters in der vorderen Hälfte des zweiten Maschinenzyklustakts aktiviert wird. Mit diesem Signal kann man einen begonnenen Schreibzyklus noch abwenden. Ein RAM kann nicht beschrieben werden; das klingt komisch. Es wird spätestens dann sehr logisch.


Adreß-
takt
AC
DS
WE
CS
AØ
A1
A2
A3
A4
A5
A6
A7
A8
A9
A1Ø
A11
Z6132
4Kx8
BUSY
UBB
UCC
Null
DØ D1 D2 D3 D4 D5 D6 D7
Adreß-
Leitungen
(statisch)
Refresh-Betriebsart
Definition
(L0 ≙ Eigenrefresh)
Substrat-
Auskopplung
0,1µF
+5V
Masse
Daten
(bidirektional)

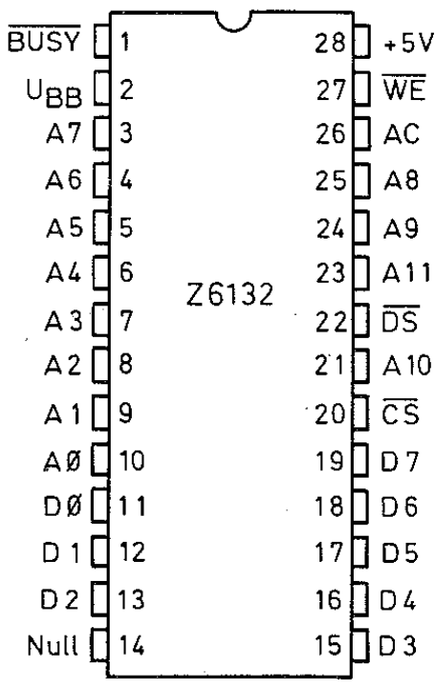

BUSY 1
UBB 2
A7 3
A6 4
A5 5
A4 6
A3 7
A2 8
A1 9
AØ 10
DØ 11
D1 12
D2 13
Null 14
Z6132
28 +5V
27 WE
26 AC
25 A8
24 A9
23 A11
22 DS
21 A10
20 CS
19 D7
18 D6
17 D5
16 D4
15 D3

Bild 4.22

4 Phasen der internen Operation:

1. Speicher lesen oder schreiben oder schreiben mit Inhibit
2. Vorbereiten und Umschalten
3. Auffrischen (aufladen) der dynamischen Transistorspeicherzellen
4. Vorbereiten und Umschalten

Funktionszyklus-Diagramm

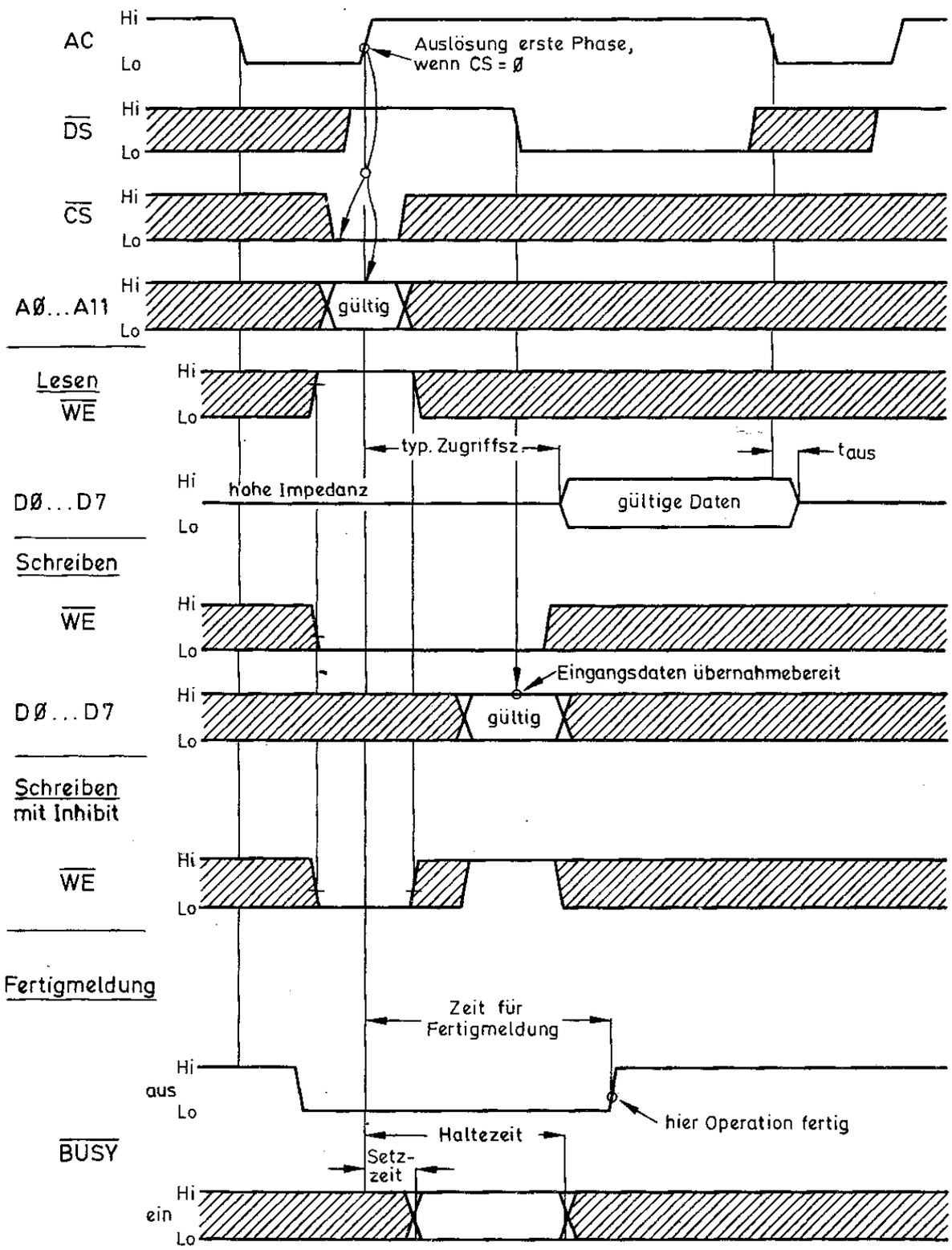


Bild 4.23

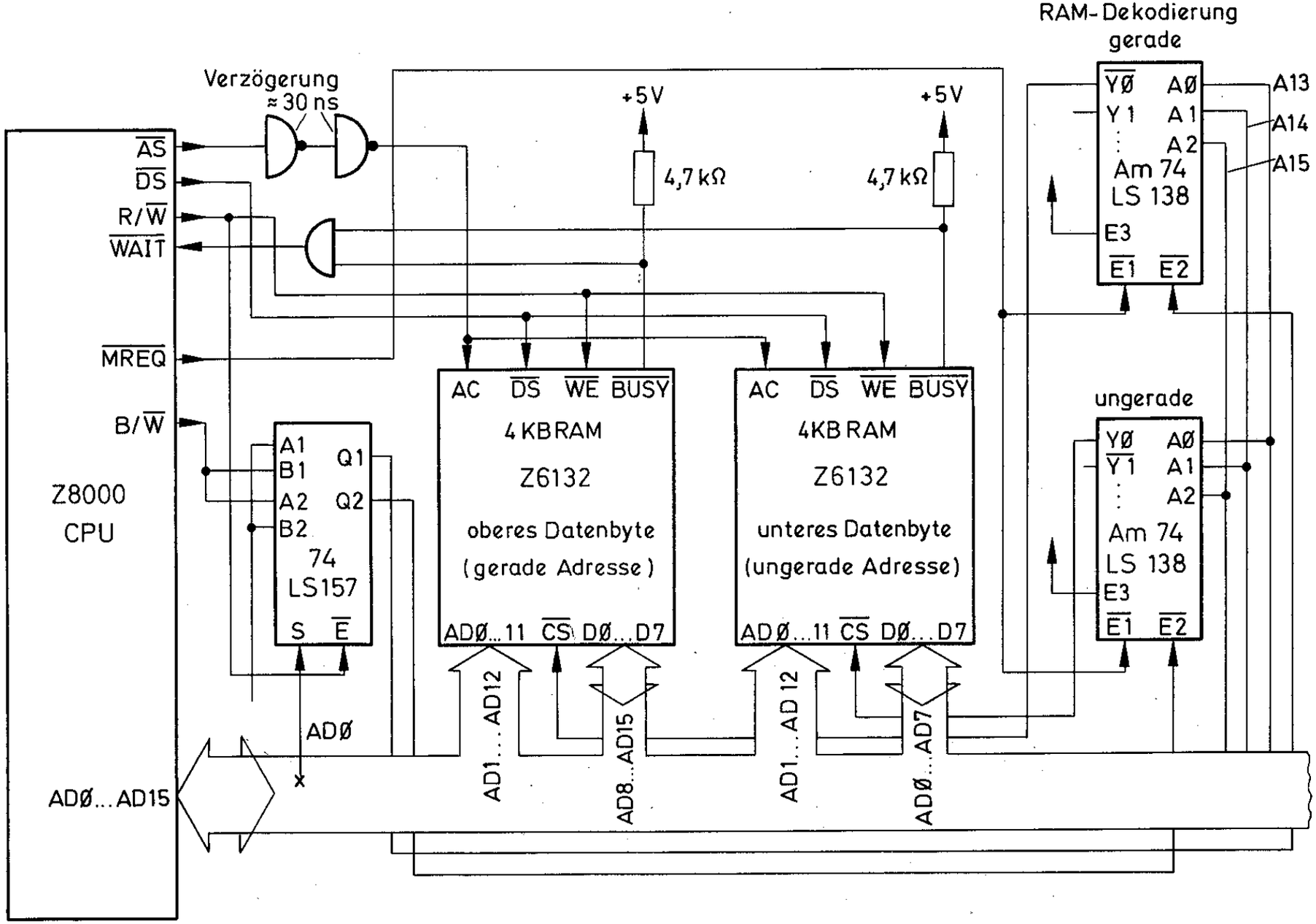

RAM-Dekodierung
gerade
ungerade
Verzögerung
≈30 ns
+5V
+5V
4,7 kΩ
4,7 kΩ
AS
DS
R/W
WAIT
MREQ
B/W
Z8000
CPU
A1
B1
A2
B2
Q1
Q2
74
LS157
S
E
AC
DS
WE
BUSY
4 KB RAM
Z6132
oberes Datenbyte
(gerade Adresse)
unteres Datenbyte
(ungerade Adresse)
AD0...11
CS
D0...D7
YØ
Y1
AØ
A1
A2
Am 74
LS 138
E3
E1
E2
A13
A14
A15
ADØ
ADØ...AD15
AD1...AD12
AD8...AD15
AD0...AD7

Bild 4.24

wenn man bedenkt, daß wichtige Daten überschrieben würden, könnte man dies nicht mit $\overline{SUP}$ der MMU verhindern. Das Segment kann erst dann wieder überschrieben werden, wenn im Beschreibungsregister für dieses Segment in der MMU eine neue Definition gewählt wird. Bei den Zilog-Systembausteinen paßt wirklich alles zusammen. Irgendwie ist das schon faszinierend.

In **Bild 4.24** ist ein Speicherwerk in der Größe 4 K Worte dargestellt, das mit zwei Z 6132 aufzubauen ist. Adressen und Daten benötigen keine Separierung. Die Adreßdekoder sind mit einfachen LS 138 zu bilden, weil die Selektadreßleitungen nicht gespeichert werden müssen. $\overline{CS}$ aktiv reicht für die Zeit, in der die Adresse am Bus anliegt. Die Umschaltung zwischen Byte- und Wortzugriff wird ebenfalls sehr einfach gemacht. Ein Vierfach-Datenselektor LS 157 erledigt das und steuert mit seinen Q-Ausgängen sogleich die Bedingungseingänge $\overline{E2}$ der Adreßdekoder an (gerade und ungerade). $\overline{MREQ}$ gleich Lo ist die erste Bedingung an den $\overline{E1}$-Eingängen. An den $\overline{BUSY}$-Ausgängen der Speicher wird durch die UND-Verknüpfung jedes Lo auf den $\overline{WAIT}$-Eingang der CPU gegeben, das bei jedem Eigenrefresh ausgelöst wird.

In **Bild 4.25** und **4.26** sind die beiden quasi statischen Speicher noch einmal mit der direkten Verbindung zum AD-Bus dargestellt. Beim 4816-2 muß das invertierte Signal von $\overline{AS}$ verwendet werden, um $\overline{CE}$ zu aktivieren. CS muß gleichzeitig oder früher aktiviert werden. OE muß unbedingt aus den CPU-Signalen $\overline{DS}$ und R/$\overline{W}$ gebildet werden. Für den

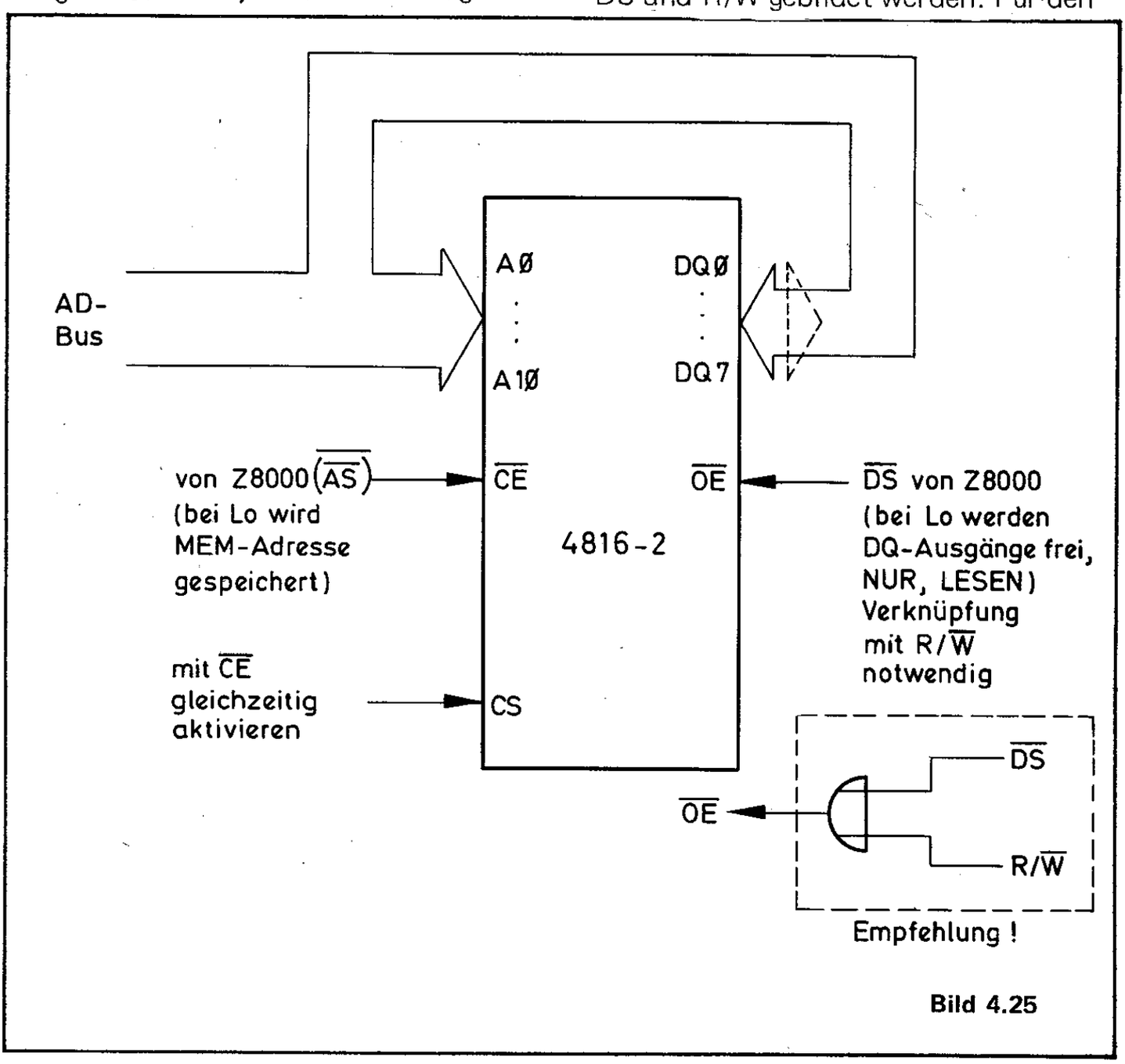


Bild 4.25

Z 6132 sollte eigentlich nur beachtet werden, daß $\overline{AS}$ der CPU um etwa 30 ns verzögert wird, das etwa der Laufzeit der Adreßdekodierung im LS 138 entspricht. Dies sollte man ernst nehmen, weil ansonsten die Lo ↑ Hi-Flanke an $\overline{AS}$ früher kommt, bevor $\overline{CS}$ aktiviert ist, — und das darf nicht sein.

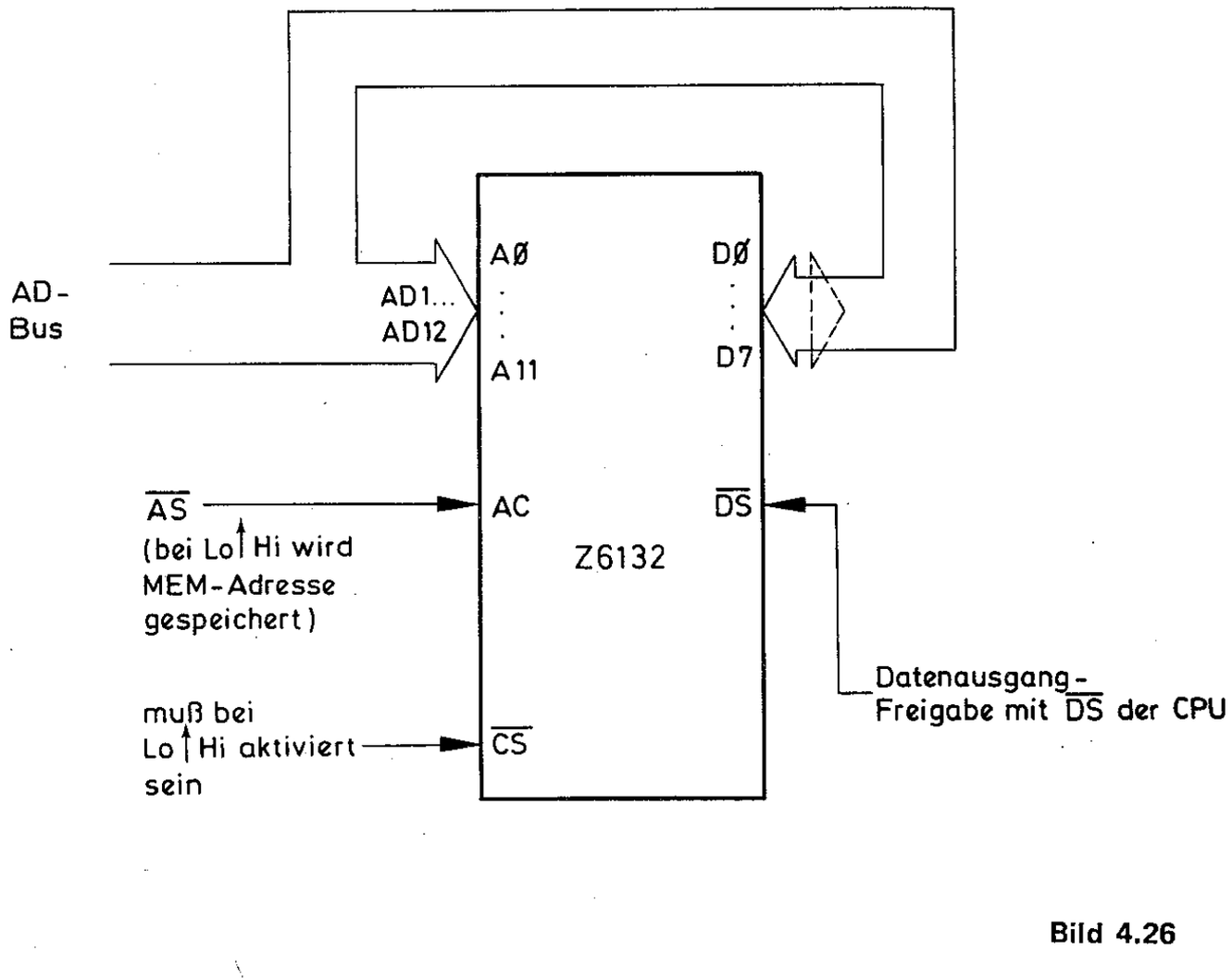
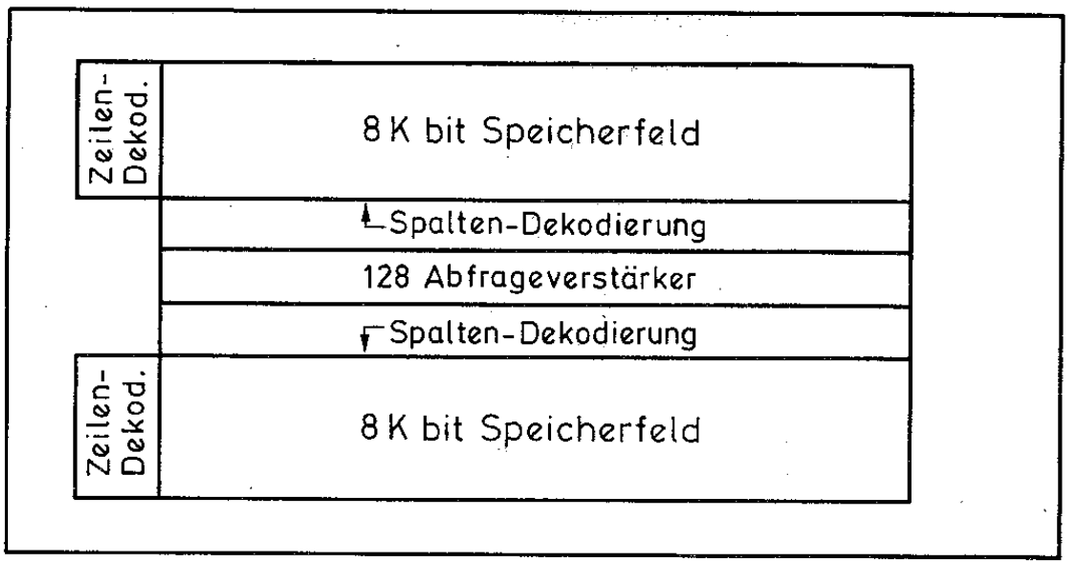


Bild 4.26

4.4 ANSCHLUSS-SCHEMATIK FÜR DYNAMISCHE SCHREIB-/ LESESPEICHER

Die Z 8000-CPU ist mit einem Refresh-Zählerregister ausgestattet, das es sehr einfach gestaltet, einen Arbeitsspeicher aus dynamischen RAMs zu betreiben. In Abschnitt 2.5 wurde die Funktion dieses Registers erklärt. Dynamische RAMs haben in ihrer Quantität deutlich zugenommen (Zeitraum 1977 bis 1980), weil sie im allgemeinen eine Reihe von Vorteilen gegenüber statischen RAM-Bausteinen aufweisen. Für das Jahr 1979 kann das 16K x 1 Bit dynamische RAM im 16poligen DIL-Gehäuse als generell weitverbreiteter Industriestandard gelten. Der Preisverfall dieses Typs gab im übrigen einen weiteren Anstoß für einen zunehmenden Einsatz. Es steht außer Frage, daß dieser Speichertyp mit seinen vielfältigen Bezeichnungen (2116, 4116, 6116, 8116, 9016, u416, F16K, TMS 4070) auch noch in den 80er Jahren in großen Stückzahlen eingesetzt werden wird. Sicher werden dann die "single supply"-Typen mit nur +5 V stark auf



Mostek 4116 Chip - Architektur
(nicht vergleichbar mit Intel 2116)

Bild 4.27

dem Vormarsch sein, doch das Format 16K x 1 wird noch lange Lebensdauer erwarten dürfen. Wäre es auch anders zu erwarten, als daß diesem Typ im Kapitel über Speicheranschlüsse an den Z 8000 eine zentrale Bedeutung beigemessen wird. Wurde man im Abschnitt zuvor mit den Eigenschaften und Anwendungen des Z 6132 geradezu verwöhnt, steht für diesen Typ dem 16K x 1 ein sehr "grimmiger" Preis gegenüber. Für einen 6132 bekommt man etwa 16K Worte (16 Bausteine).

Der Chip eines dynamischen 16K-Speichers besitzt nach der strukturellen Anordnung der Version von Mostek zwei 8 K-Speicherfelder mit je 128 Zeilen und 64 Spalten. An den Schnittpunkten befinden sich jeweils ein NMOS-Transistor mit einer Kapazität von etwa 0,04 pF ($= 4 \cdot 10^{-14}$ Farad) in der Drainleitung. Dieser winzige Kondensator ist die eigentliche Speicherkapazität für digitale Informationen (das dynamische des Speicherbausteins). Er liegt mit einem Ende am Drainanschluß des FET und am anderen Ende auf +12 V. **Bild 4.27** zeigt die Gliederung der Funktionsgruppen (Speicherfelder, Dekodierung, Abfrageverstärker) auf einem 16K-Chip. In **Bild 4.28** wird ein Abschnitt des Speicherfeldes mit seinen Speicherzellen und einem dynamischen Abfrageverstärker dargestellt. Die Speicherung einer digitalen Null oder Eins in einen Kondensator in Form eines Lo- oder Hi-Spannungspegels unterliegt dem physikalischen Umstand, daß der Spannungskegel nach einer e-Funktion $u(t) = U \cdot e^{-dt}$ fällt. Die Selbstentladung auf Grund von Leckströmen ist unaufhaltsam. Deshalb müssen die Speicherzellen ständig abgefragt werden, bevor sie ihre Ladung ganz verlieren. Nach der Abfrage werden sie wieder auf ihren momentanen Spannungspegel Hi oder Lo aufgeladen, sodaß die gespeicherte Information erhalten bleibt. Diesen Vorgang nennt man allgemein üblich "Refresh" oder auffrischen, der für alle Zellen zyklisch innerhalb von 2 ms erledigt sein muß.

In der Kapazität C1 ist beispielsweise

eine digitale Eins gespeichert. Der Transistor T1 stellt die Verbindung zur Abfrageleitung (auch Spaltenleitung genannt) her, die zum Abfrageverstärker führt. Der Gate-Anschluß von T1, wie auch die aller anderen Gates anderer Spalten, sind mit einer Zeilenleitung verbunden (Zeile 0). Wird diese Zeile vom Zeilenadreßdekoder auf Grund einer entsprechenden Adresse aktiviert, wird neben den anderen Gates auch das Gate von T1 angesteuert, so daß die Eins-Ladung von C1 auf den Abfrageverstärker geführt und entsprechend verstärkt werden kann (erste Phase). Wird in der zweiten Phase auf Grund einer dekodierten Spaltenadresse der Transistor bei A, der Spalten- und Daten-E/A-Leitung trennt, zum Durchschalten gebracht, wird damit die Information "Eins" aus C1 auf den Ausgang des Bausteins freigegeben. Somit ist damit der Lesevorgang und die Bedeutung der Zeilen- und Spaltenadresse bei dynamischen RAMs angedeutet.

In der ersten Phase beim Durchschalten des Transistors T1 auf die Spaltenleitung wird zugleich die Ladung in C1 damit aufgefrischt. Der Verstärkungseffekt des Abfrageverstärkers hat in erster Linie diesen Zweck. Hiermit erklärt sich jetzt auch der Auffrischvorgang bei dynamischen RAMs. Zum Auffrischen aller Speicherzellen ist nur eine Zeilenadresse notwendig. Zuerst wird Zeile 0 abgefragt und aufgeladen, dann Zeile 1, 2 usw. Es werden alle Speicherzellen der Zeile 0, 1 usw. jeweils gemeinsam aufgefrischt. Eine Zeilenadresse bewirkt die Aufladung von 2 x 64 Zellen. Ein Auffrischzähler erfordert somit die Länge von 7 bit für 128 Zeilenadressen. Ein Auffrischadreßzähler gibt eine 7-bit-Adresse ab und inkrementiert nach erfolgtem Refresh einer Zeile in einem Feld seinen Wert um 1. Der Z 8000 besitzt in seinem Refreshregister einen 9-bit-Zeilenadreßzähler, damit können nicht nur die 16K-Typen, sondern auch die 64K x 1 dyn. RAMs und künftig auch einmal kommende 256K x 1 dyn. RAMs aufgefrischt werden. 64K-Typen benötigen 256 Zeilenadressen, 256K-Typen werden 512 Zeilenadressen benötigen. Es gibt übrigens zwei Typen von Abfrageverstärkern; den dynamischen Typ, wie er in Bild 4.28 gezeigt ist, und den statischen Typ. Der Abfrageverstärker hat prinzipiell die Aufgabe, winzige Ladungsströme zu verstärken, — und das im Subnanosekundenbereich. Dynamische Abfrageverstärker stellen sehr hohe Anforderungen an das Layout und das Design von dynamischen RAM-Chips. Eine Chiparchitektur mit dynamischen Abfrageverstärkern bringt allerdings eine erhebliche Reduzierung des Stromverbrauchs und bietet zudem sehr kurze Zugriffszeiten. 100 ns sind in jedem Fall möglich. Die Technik mit dynamischen Abfrageverstärkern wird nicht von allen Herstellern angewandt. Die Typen MK 4116 von Mostek, sowie des Z 6116 von Zilog und der Am 9016 von AMD arbeiten mit dynamischen Abfrageverstärkern. Man erkennt sie daran, indem sie für U_{DD} +12 V nur den maximalen Strom von I_{DD} = 35 mA (aktiv) benötigen. Der Typ 2116 mit statischen Abfrageverstärkern benötigt bei U_{DD} (+12V) einen maximalen Strom von I_{DD} = 69 mA.

In **Bild 4.29** wird das 16K x 1 Bit dynamische RAM signaltechnisch vorgestellt. Es ist nicht so schwer zu verstehen, im Gegensatz zu manch anderen Meinungen. $\overline{RAS}$ und $\overline{CAS}$ sind die entscheidenden Adreßsteuersignale für diesen Speicherbaustein. Diese Lo-aktiven Eingänge sind zugleich als Freigabeeingänge zu betrachten. Folgende Permutationen von $\overline{RAS}$ und $\overline{CAS}$ sind möglich:

- $\overline{RAS}$ und $\overline{CAS}$ → Hi: Baustein nicht aktiv
- $\overline{RAS}$ → Lo. $\overline{CAS}$ → Hi: Baustein aktiv erste Phase lesen/schreiben oder Refresh, Ausgang D_{out} nicht aktiv (hochohmig)
- $\overline{RAS}$ → Lo, $\overline{CAS}$ → Lo: Baustein aktiv, zweite Phase lesen/schreiben, Ein Ein-/Ausgabe
- $\overline{RAS}$ → Hi, $\overline{CAS}$ → Lo: Baustein nicht aktiv, keine Bedeutung

Speicherzellen des dynamischen RAMs 4116 von Mostek
mit dynamischen Spalten-Abfrageverstärkern
(N-Kanal-Polysilizium II-Prozess)

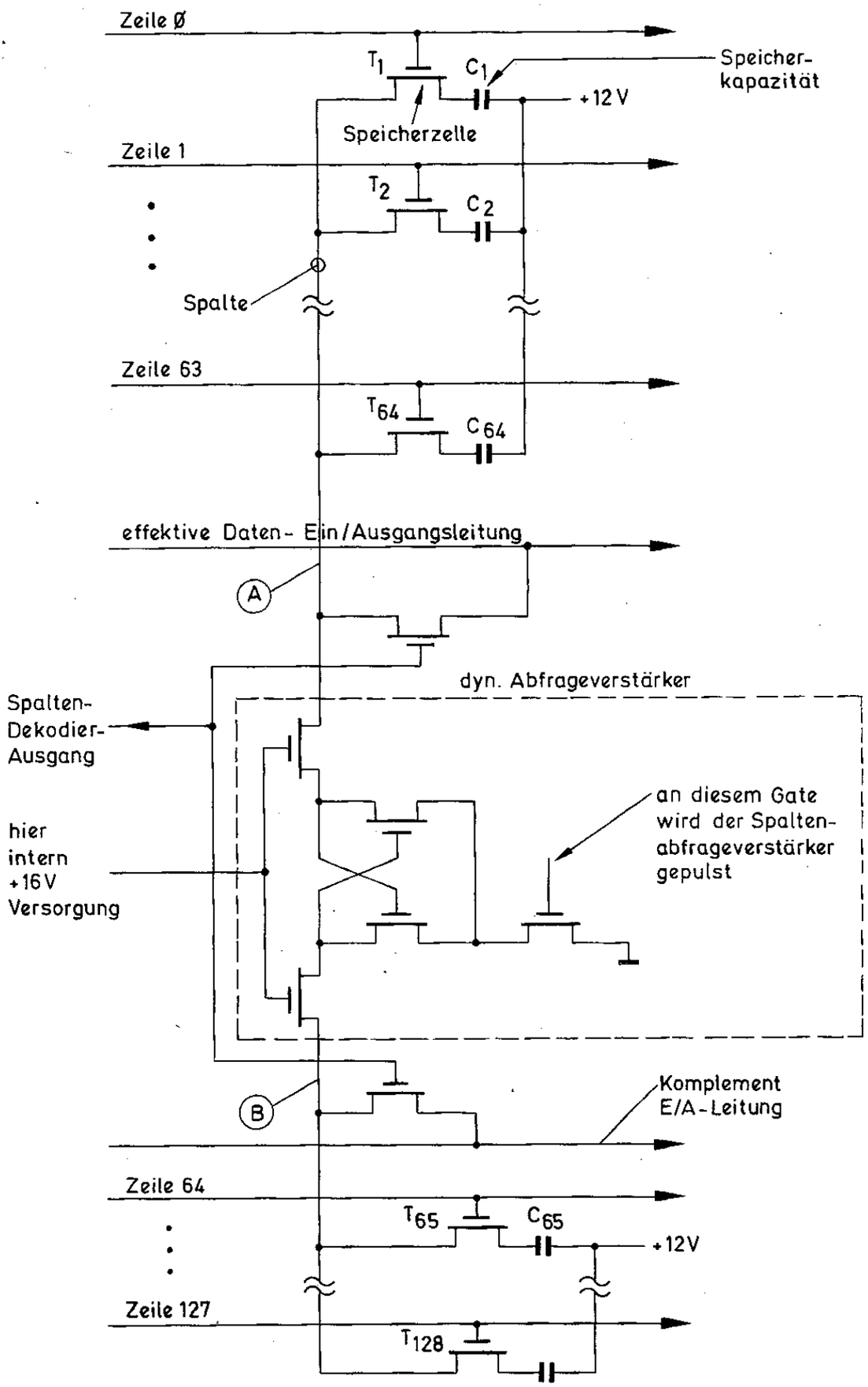


Bild 4.28

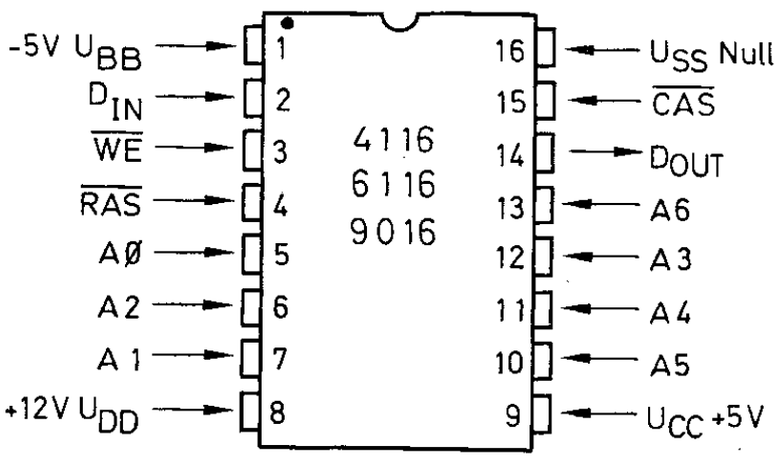


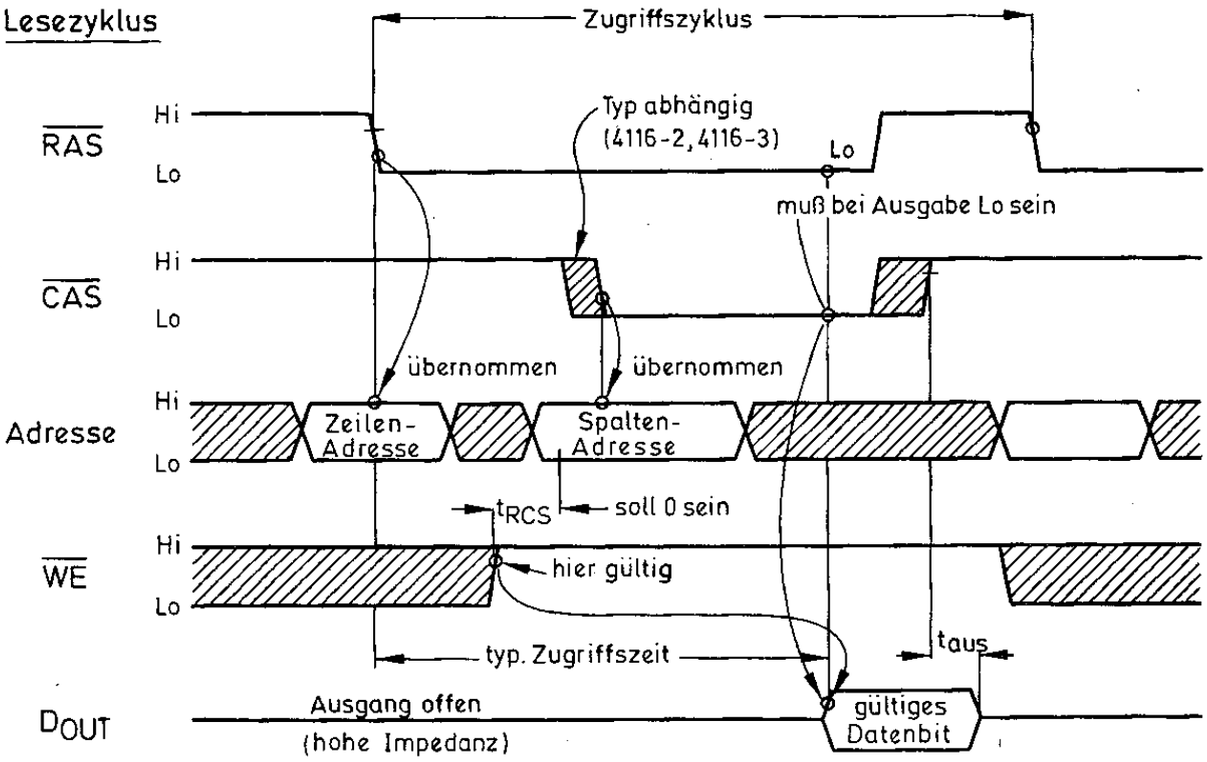
Bild 4.29

Der Ausgang D_{out} wird von $\overline{CAS}$ gesteuert, wird aber nur dann frei, wenn auch $\overline{RAS}$ aktiv ist. Mit der Hi ↓ Lo-Flanke an $\overline{RAS}$ wird ein Zyklus (erste Phase) generell eingeleitet, indem der Zeilenadreßteil von den Dekodern für die Zeilenleitungen vorbereitet und angelegt wird. Die 7-bit-Zeilenadresse muß für den schnellsten Typ 20 ns sicher anliegen, damit sie fehlerfrei übernommen wird. Sodann wird $\overline{CAS}$ nach etwa 30 . . . 40 ns aktiviert (Hi ↓ Lo-Flanke), wonach an den Adreßeingängen die Spaltenadresse sicher anliegen muß. Mit der Hi ↓ Lo-Flanke von $\overline{CAS}$ wird die zweite oder Transferphase mit dem Speicherbaustein eingeleitet. D_{out} kann nur aktiv werden, wenn $\overline{RAS}$ und $\overline{CAS}$ Lo und $\overline{WE}$ gleich Hi ist. Erinnert man sich noch einmal an das MMU-Signal $\overline{SUP}$, kann damit das Schreiben verhindert werden, indem $\overline{CAS}$ nicht auf Lo gebracht wird, auch wenn $\overline{WE}$ bereits Lo ist. In **Bild 4.30** ist der Auffrischzyklus dargestellt, der bekanntlich nur in der ersten Phase abgewickelt wird. $\overline{CAS}$ wird nicht Lo.

Mit dieser kurzen Vorstellung des 16K-x 1 dynamischen RAM ist die Vorbedingung geschaffen, die vielfältigen und teilweise komplizierten Speicherschaltungen für den Z 8000 zu verstehen, die in diesem Abschnitt beschrieben werden.

$\overline{RAS}$ = Zeilenadreßabtastung bzw. -übernahme
$\overline{CAS}$ = Spaltenadreßabtastung bzw. -übernahme

Lesezyklus



Schreibzyklus

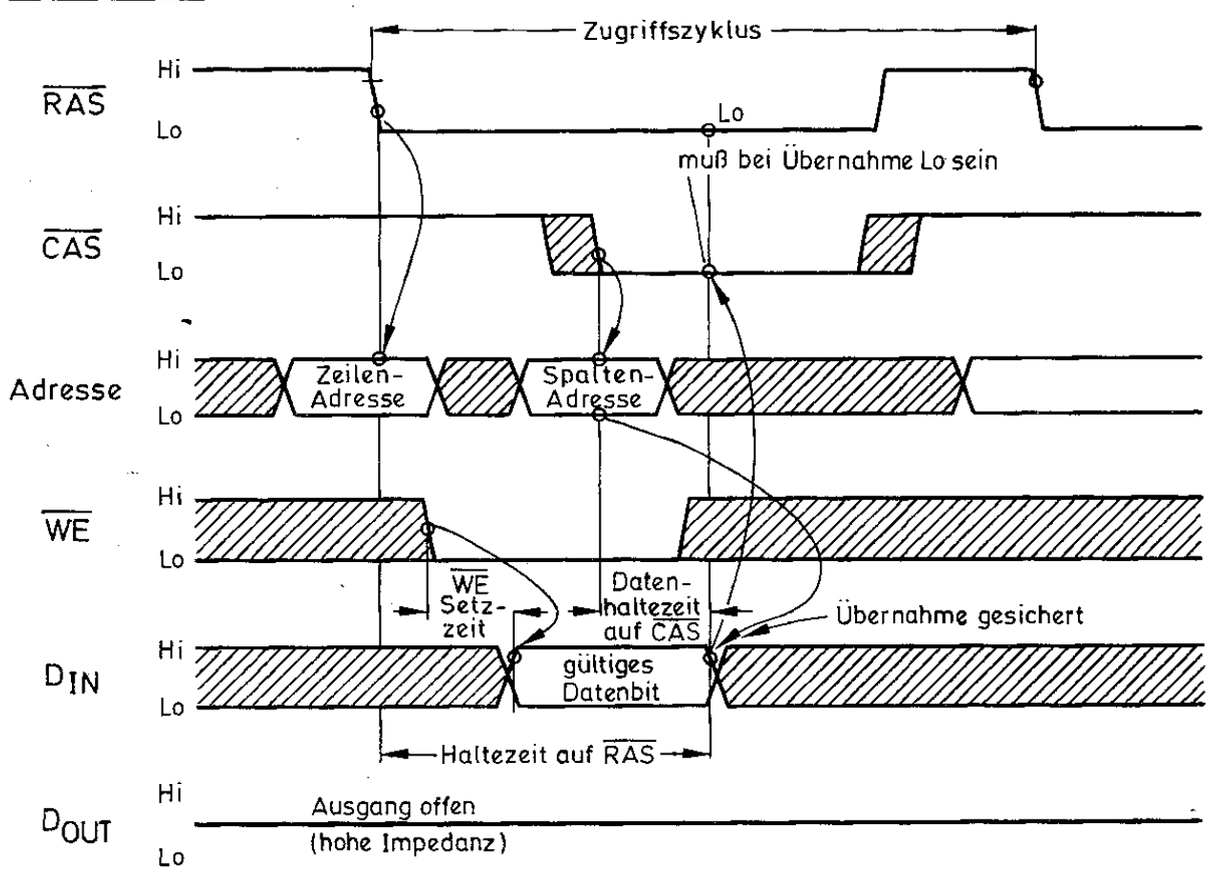


Bild 4.29

Zeilenadreß - Auffrischzyklus

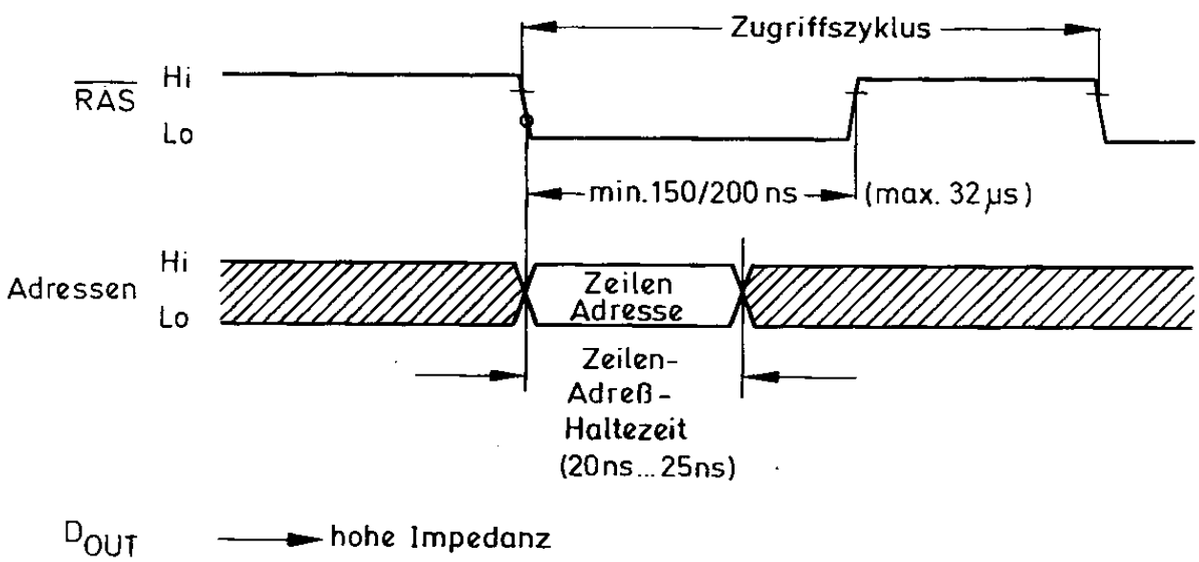


Bild 4.30

4.4.1 Interface für Spalten-/Zeilen-Adreßmultiplex

Dynamische RAMs mit interner Multiplexlogik kann man nicht adressieren wie übliche statische Typen. 16K x 1 RAMs erfordern eine absolute Adreßlänge von 14 bit. Die am 16poligen DIL-Gehäuse angebrachten Adreßeingänge A0 . . . A6 dienen zur sequentiellen Übernahme der Zeilenadresse und anschließend der Spaltenadresse. **Bild 4.31** zeigt den grundlegenden Zusammenhang zwischen statischer Speicheradressen und den Adreßeingängen dieses dynamischen RAM-Typs. Eine Multiplexlogik aus UND-/ODER-Gattern, vereinzelt auch NOR, legt die Adreßteile an die Eingänge. Die zeitliche Folge wird durch Signale an $\overline{RAS}$ und $\overline{CAS}$ festgelegt, die aus einer separaten Adreßdekodierung erzeugt werden. Diese Anordnung ist speichertypisch und nicht auf eine CPU bezogen.

Die praktische Ausführung der SchaltunG eines Speicherwerks mit 16K dynamischen RAMs zeigt **Bild 4.32.** Als Adreßmultiplexer dienen zwei Vierfach-2-bit-Datenselektoren des Typs LS 157. Ein Baustein besitzt je 4 Eingänge A und B, die in der Reihenfolge gepaart sind. Die Q-Ausgänge steuern die Adreßeingänge des RAM-Feldes an. An den S-Eingängen der Multiplexer wird von den Eingängen A auf die Eingänge B umgeschaltet. Das Umschalten von Spalte auf Zeile erfolgt sehr schnell (Differenz etwa 40 ns). An das Flipflop LS 74 wird zu diesem Zweck ein Vielfaches des Z-Taktes angelegt. Bei der Umschaltung von Zeile auf Spalte wird am unteren Multiplexer der immer auf Null liegende Eingang B4 auf $\overline{CAS}$ durchgeschaltet, wenn $\overline{MREQ}$ Lo war, also ein Speicherzugriff erfolgen soll. Ist das der Fall, wird $\overline{CAS}$ gleich Lo und die schon anliegende Spaltenadresse in die RAMs übernommen.

Die Dekodierlogik bewertet die Adreßleitung A0 auf gerade oder ungerade, das bei B/$\overline{W}$ gleich Hi wichtig ist. Dementsprechend wird das Zeilenadreß-Auswahlsignal $\overline{RAS1}$ bzw. $\overline{RAS2}$ für das obere oder untere oder für beide Bytes (B/$\overline{W}$ = Lo) erzeugt.

Die 33 Ω-Widerstände an den Adreß- und Freigabeleitungen sind unbedingt einzubauen, weil das Treiben von NMOS-Schaltungen mit Schottky-TTL-Gattern bestimmten Gesetzen unterliegt. Es spielt hier die Impedanz des LS-Treibers bei Hi oder Lo eine Rolle und ebenso die Hi ↓ Lo- oder Lo ↑ Hi-Flanke eines Signals an die MOS-Eingänge. Bedeutsam ist das negative Überschwingen bei einer Hi ↓ L0-Flanke an den Q-Ausgängen, das unbedingt so gering wie möglich sein muß. Es entsteht auf Grund der Ausgangsimpedanz des Treiberausgangs bei Lo-Pegel. Mit 33 Ω wird dieses Unterschwingen auf ein Minimum reduziert. Es gibt hierfür noch eine bessere Lösung, wie etwas später noch gezeigt wird.

Die Schaltung in Bild 4.32 ist überdies technisch nicht bis ins letzte Detail ausgeführt, weil sie von den Bausteinen als veraltet gilt, nicht zu verwechseln von der Methode. Die Selektoren LS 157 werden nur noch vereinzelt verwendet und die Widerstände sind nicht gerade vorteilhaft. Im folgenden wird weit besseres geboten.

Ein erster Schritt zu einer Vereinfachung des Multiplex-Interfaces stellt der Baustein 3242 dar, der von Intel und anderen Herstellern angeboten wird. Dieser Adreßmultiplexer mit eigenem Refreshzähler ist speziell auf die dynamischen 16K-RAMs zugeschnitten. Der 28polige Baustein (Innenaufbau **Bild 4.33**) hat 14 Adreßeingänge A0 . . . A13 und 7 Multi-

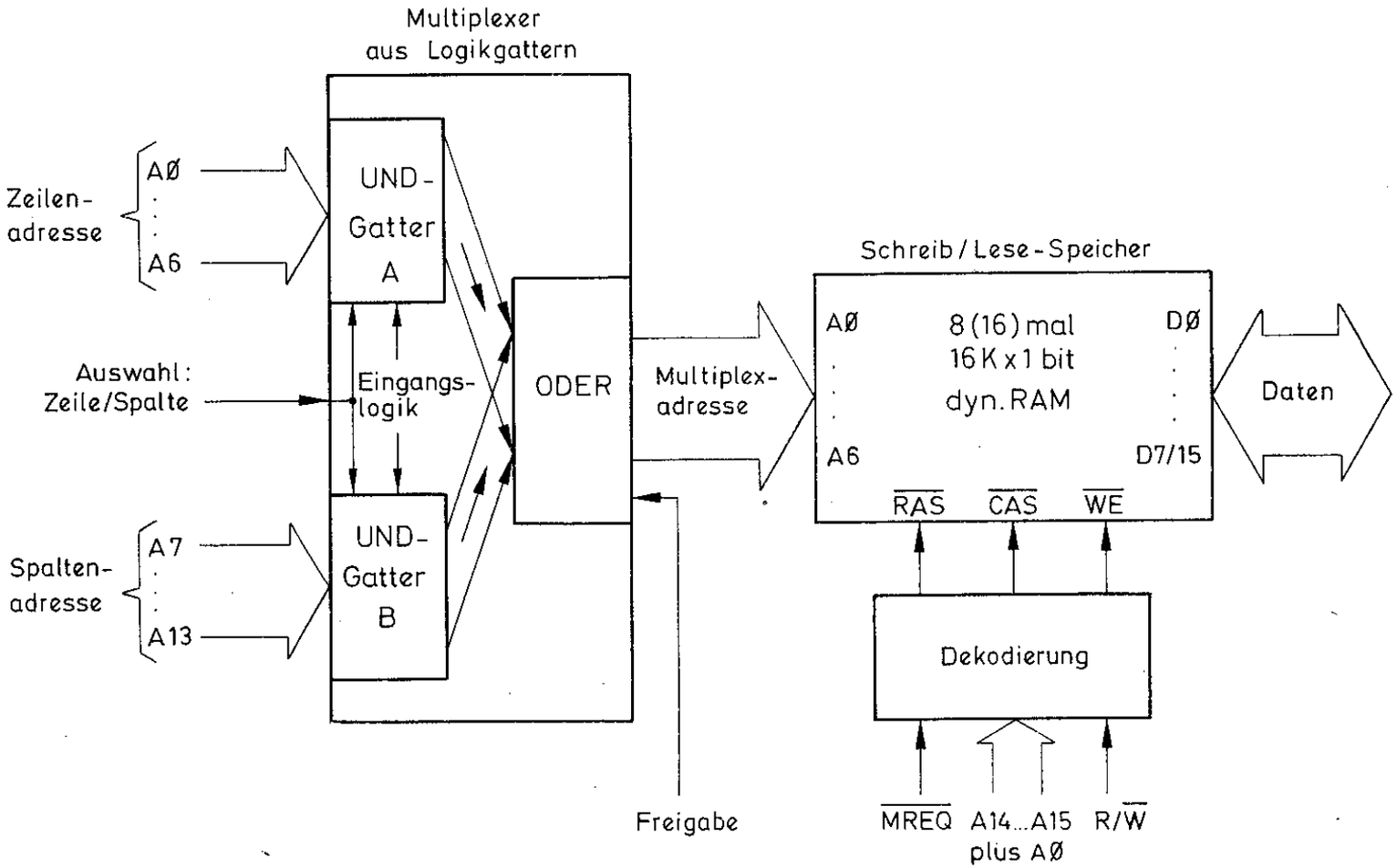

Multiplexer
aus Logikgattern
Zeilen-
adresse
AØ
A6
UND-
Gatter
A
Auswahl:
Zeile/Spalte
Eingangs-
logik
Spalten-
adresse
A7
A13
UND-
Gatter
B
ODER
Multiplex-
adresse
Freigabe
Schreib / Lese-Speicher
AØ
A6
8 (16) mal
16 K x 1 bit
dyn. RAM
DØ
D7/15
Daten
RAS
CAS
WE
Dekodierung
MREQ
A14...A15
plus AØ
R/W

Bild 4.31

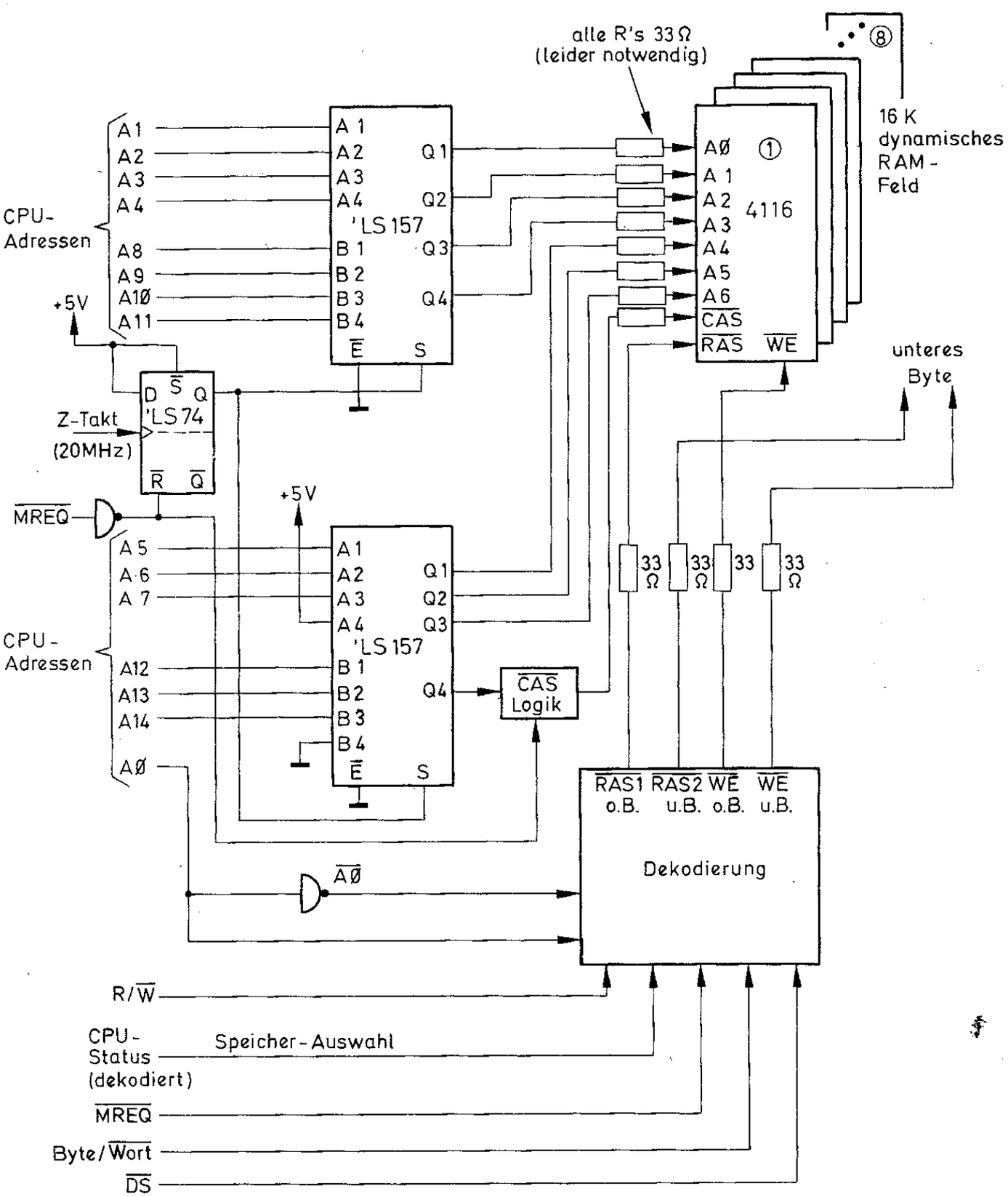

alle R's 33 Ω
(leider notwendig)
⑧
16 K
dynamisches
RAM-
Feld
A1
A2
A3
A4
CPU-
Adressen
A8
A9
A10
A11
A 1
A2
A3
A4
'LS 157
B 1
B 2
B 3
B 4
Q 1
Q2
Q3
Q4
Ē
S
+5V
D
S̄
Q
'LS 74
Z-Takt
(20MHz)
R̄
Q̄
AØ
A 1
A2
A3
A4
A5
A6
CAS
RAS
WE
4116
①
unteres
Byte
MREQ
+5V
A 5
A 6
A 7
CPU-
Adressen
A12
A13
A14
AØ
33 Ω
33 Ω
33
33 Ω
CAS
Logik
RAS1 o.B.
RAS2 u.B.
WE o.B.
WE u.B.
Dekodierung
AØ
R/W
CPU-
Status
(dekodiert)
Speicher-Auswahl
MREQ
Byte/Wort
DS

Bild 4.32

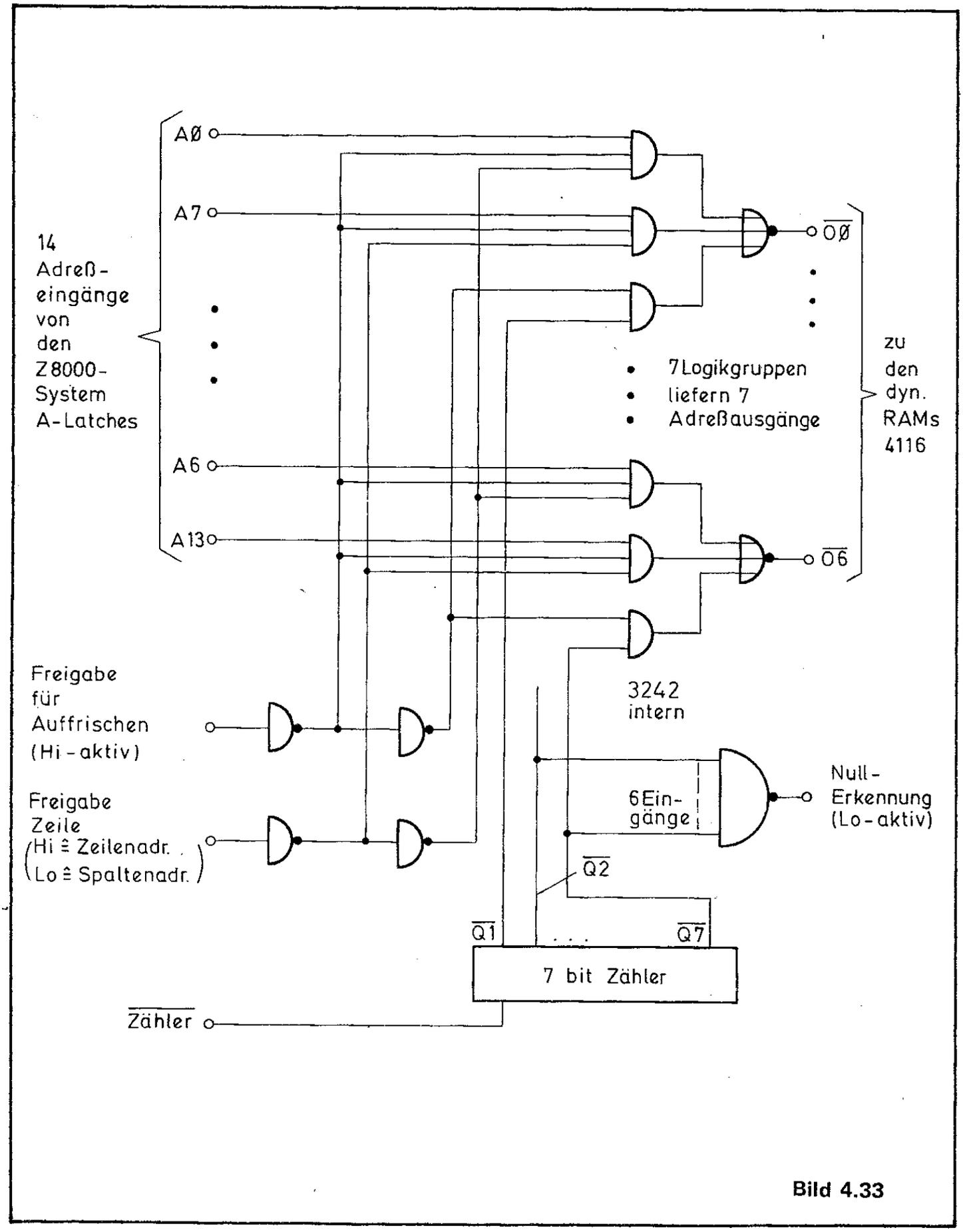


Bild 4.33

plexausgänge aus NOR-Gattern, die die Zeilen- oder Spaltenadresse invertiert ausgeben. An dem Steuereingang "Freigabe Zeile" werden bei Hi die Adreßeingänge A0 . . . A6 auf $\overline{Q0}$ . . . $\overline{Q6}$ durchgeschaltet und bei Lo die Eingänge A7 . . . A13. Die Ausgänge des 7-bit-Adreßzählers für eine Refreshadresse werden auf $\overline{Q0}$ . . . $\overline{Q6}$ gegeben, wenn am Eingang "Freigabe für Auffrischen" ein Hi anliegt. Da diese Funktion für ein Z 8000-Speicherwerk nicht benötigt wird, kann man diesen Eingang immer auf Lo legen, weil die Refreshadresse

immer von der CPU über A eingegeben wird. **Bild 4.34** zeigt das Anschlußbild dieses Adreßmultiplexers. Die Multiplexausgänge sind an den unteren Anschlüssen des Bausteins zusammengefaßt, so daß hiermit auch eine Hilfestellung bei der Erstellung eines Leiterplatten-Layouts gegeben wird. Die Spezialisten wissen um den kritischen Verlauf der Verdrahtung dynamischer Speicherwerke. Das Beispiel mit dem 3242 zeigt hier, wie konsequent kurz die Leiterbahnen zu verlegen sind. Die Speicherbausteine sind auf engstem Raum nebeneinander anzuordnen; die Widerstände auf kürzeste Distanz zu legen.

Wie schon angedeutet, wird statt der Widerstände eine bessere Möglichkeit geboten. AMD bietet als Treiberbausteine die Typen AmZ 8165 und AmZ 8166 an, deren Ausgänge unmittelbar zum Anschluß an die Adreßeingänge der RAMs geeignet sind. **Bild 4.35** zeigt den Baustein im 20poligen DIL-Gehäuse und zugleich ein Beispiel der Verlegung des Layouts. Der Ausschnitt der Schaltung eines Ausgangstreibers läßt erkennen, warum extern keine Widerstände mehr nötig sind. Das Überschwingen in den negativen Bereich bei einer Hi ↓ Lo-Flanke wird fast eliminiert. Dafür sorgt die symmetrische Auslegung der Aus-

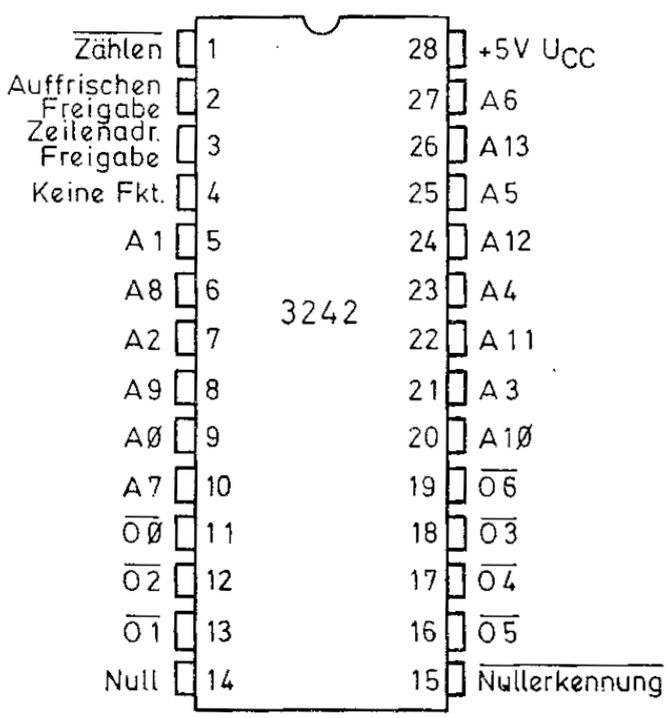
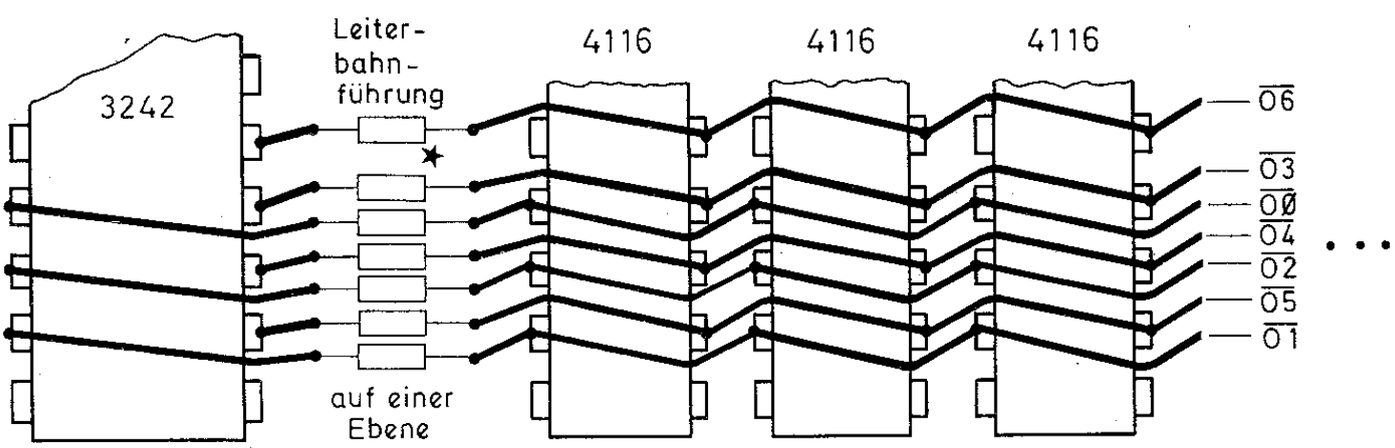


Bild 4.34

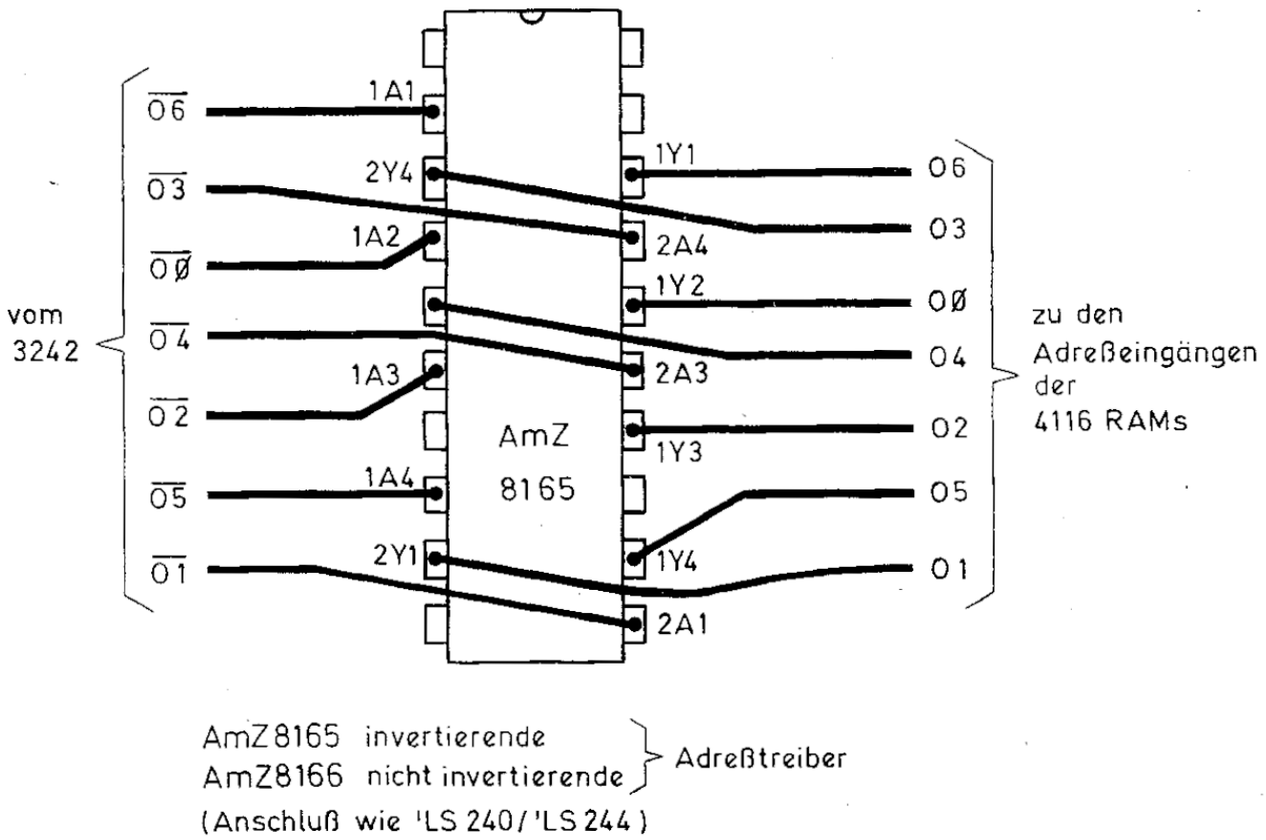


★alle Widerstände 33 Ω

Bild 4.34

gangsstufe mit Widerständen. Eine Ausgangsbelastung von 35 mA ist möglich, sowohl Quell- als auch Senkstrom. Die Verzögerung bei einer Lo ↑ Hi-Flanke beträgt typisch 9 ns bzw. 22 ns, bei einer Hi ↓ Lo-Flanke 12 ns bzw. 30 ns. Die geringere Zeit weist sich durch eine kapazitive Ausgangsbelastung von 50 pF, die längere Verzögerungszeit mit 500 pF aus. Ohne kapazitive Last ist die Verzögerung mit weniger als 10 ns angegeben, was jedoch der Realität nicht entspricht.

Mit dem genannten Adreßmultiplexer und den direkten Adreßtreibern läßt sich ein dynamisches Speicherwerk aufbauen, das technisch relativ gut überschaubar ist.



AmZ8165 invertierende
AmZ8166 nicht invertierende
(Anschluß wie 'LS 240/'LS 244)
} Adreßtreiber

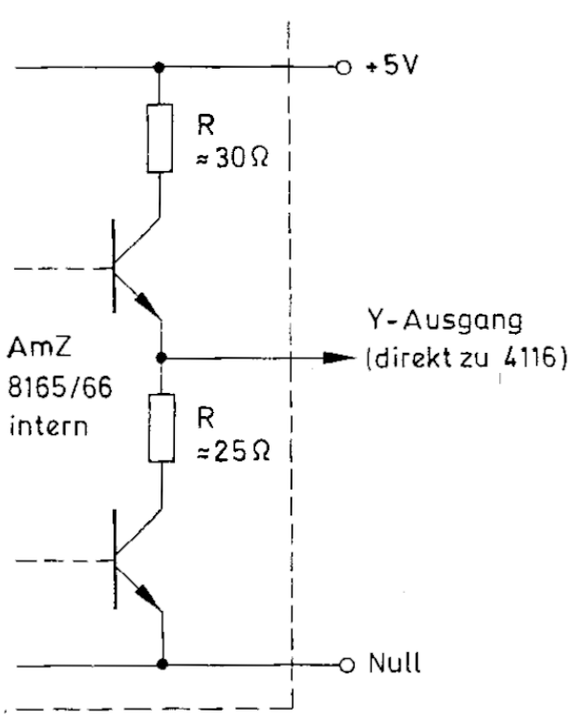


Bild 4.35

4.4.2 Beispiel einer 64 KB-Speicherbank mit 16K x 1 dynamischen RAMs

In diesem Abschnitt werden zwei Schaltungsversionen eines Interfaces für 64 64 KB dynamische Speicherbänke vorgestellt, die auf die Hardware des Z 8000 zugeschnitten sind. Speicherfelder dieser Größe gelten sowohl für den Z 8001 als auch für seinen kleineren Bruder als Standard. Sie sind als Datenfeldspeicher verwendbar oder als Programmspeicher, wenn größere Programme von Disks in den RAM geladen und mit diesem auch betrieben werden. Eine Programmausführung vom RAM aus ist immer noch am schnellsten, weil bei allgemein 100 . . . 150 ns Zugriffszeit der Bausteine keine Wartezyklen der CPU erforderlich sind.

In **Bild 4.36** ist die Schaltung einer 64 KB-Speicherbank mit kompletter Adreßmultiplex-, Treiber- und Dekodierlogik dargestellt. Die Dekodierung erfolgt nach gerader oder ungerader Adresse mit entsprechendem Transfer des oberen oder unteren Byte. Insgesamt vier 16 KB-Speichergruppen mit je 8 Bausteinen bilden das gesamte Feld. Die Ausgänge des Adreßmultiplexers 3242 werden auf die A-Eingänge des invertierenden Adreßtreibers AmZ 8165 gegeben; invertierend deshalb, weil die Adresse dann wieder dual zu den Adreßeingängen des Multiplexers ist. Von seinen Y-Ausgängen werden die Adreßeingänge aller vier Speichergruppen direkt angesteuert. Als Adreßdekoder zur Erzeugung von vier RAS-Leitungen wird der Baustein AmZ 8148 verwendet. Von den Auswahleingängen S0 . . . S2 wird S0 mit der Adreßleitung A0 und S1 mit der Adreßleitung A15 belegt. S2 wird auf Null gesetzt. $\overline{Y0}$ . . . $\overline{Y3}$ sind die Ausgänge für vier Freigabesignale einer Zeilenadreßübernahme jeweils einer 16 KB-Gruppe.

A0	A15	aktiv	Bereich	
0	0	RAS 0	0 . . . 16 KB	oberes Byte
0	1	RAS 2	17 . . . 32 KB	
1	0	RAS 1	0 . . . 16 KB	unteres Byte
1	1	RAS 3	17 . . . 32 KB	

Alle RAS-Leitungen werden über einen nicht invertierenden Treiber AmZ 8166 an die jeweiligen RAS-Eingänge der Speicherbausteine angelegt; jede RAS-Leitung an die 8 Bausteine einer Gruppe. Am Freigabeeingang des 3242 wird normal immer Hi anliegen, solange keine RAS-Leitung aktiviert ist. Das bedeutet, daß immer zuerst der Adreßteil A1 . . . A7 (Zeilenadresse von der CPU Durchlaß an die Speicherbausteine hat. Erst nachdem eine RAS-Leitung aktiviert wurde, wird der Ausgang des UND-Gatters LS 21 Null bzw. Lo werden. Hat dieses Lo eine Verzögerungszeit von etwa 50 ns durchlaufen, gelangt dieses Lo an den Freigabeeingang und schaltet den Adreßteil A8 . . . A14 (Spaltenadresse) durch. Zwischenzeitlich wurde die Zeilenadresse von den Speichern übernommen. Das Signal für die Spaltenadreßübernahme ($\overline{CAS}$) wird aus dem CPU-Signal $\overline{DS}$ gewonnen. Die Adresse liegt nach wie vor noch an den Multiplexeingängen A0 . . . A13 an und die Spaltenadresse an den Speicheradreßeingängen. Eine Speichergruppe ist bereits durch $\overline{RAS}$ aktiviert. $\overline{DS}$ wird zu Beginn des zweiten Maschinentakts aktiv. Die Daten müssen erst im dritten zur Verfügung stehen. Zwischen $\overline{CAS}$ aktiv und Datenausgabe vergehen im Speicher weniger als 100 ns. Das reicht leicht aus, ohne daß ein zusätzlicher WAIT notwendig wird. Das $\overline{CAS}$-Signal wird am $\overline{ACK}$-Ausgang (offener Kollektor, Anerkennung) des AmZ 8148 ausgegeben, wenn alle Bedingungen erfüllt sind, d.h. Status, $\overline{MREQ}$ und B/$\overline{W}$ (= Hi) aktiv. $\overline{ACK}$ wird in den Treiber AmZ 8166 gegeben und von vier

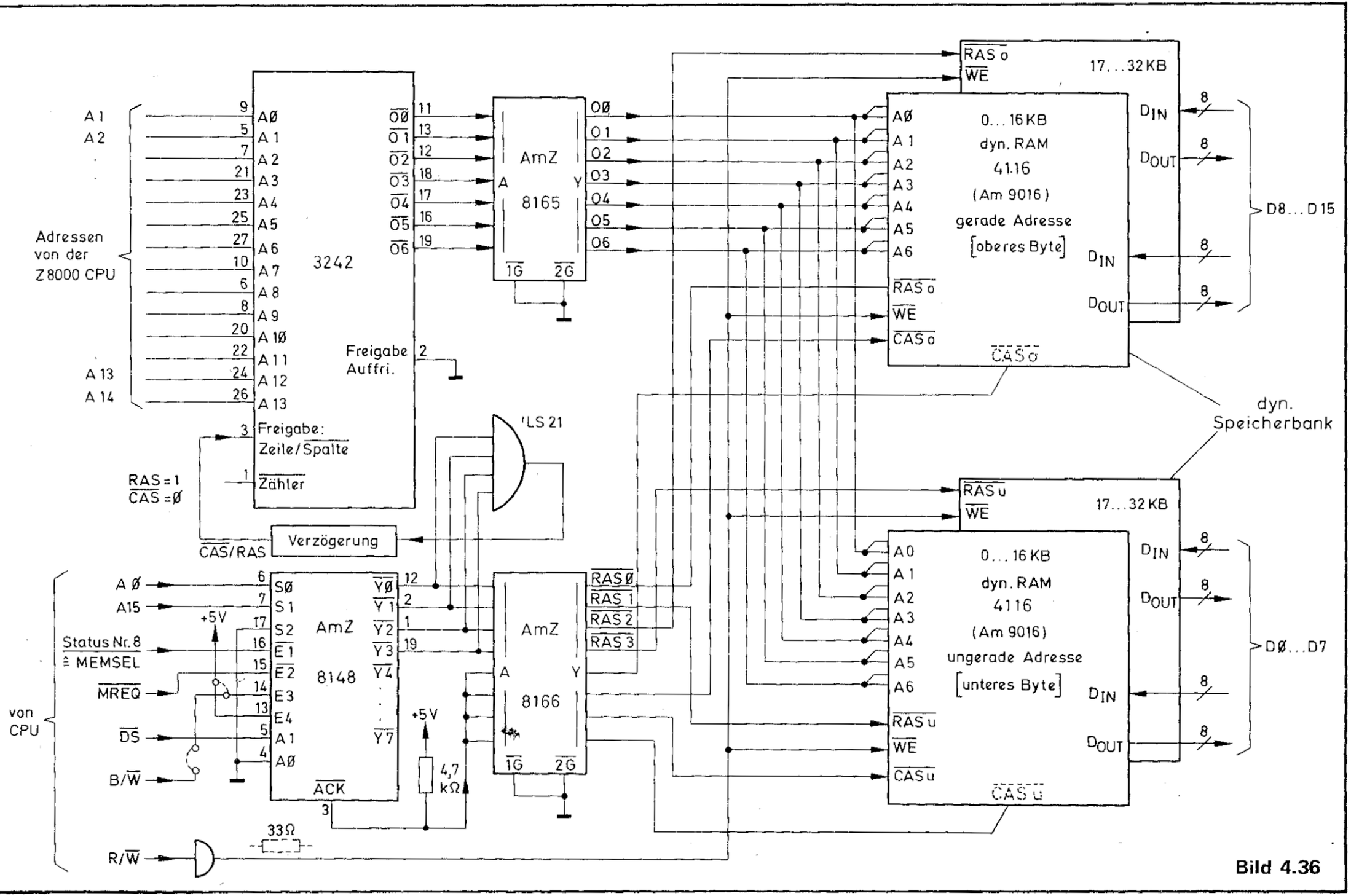

A 1
A 2
A 13
A 14
Adressen von der Z8000 CPU
3242
Freigabe Auffri.
Freigabe: Zeile/Spalte
Zähler
RAS = 1
CAS = Ø
CAS/RAS
Verzögerung
'LS 21
AmZ 8165
AmZ 8148
AmZ 8166
ACK
4,7 kΩ
33Ω
+5V
Status Nr. 8 ≙ MEMSEL
MREQ
DS
B/W
R/W
von CPU
RAS o
WE
CAS o
RAS u
CAS u
17...32 KB
0...16 KB
dyn. RAM
4116
(Am 9016)
gerade Adresse
[oberes Byte]
ungerade Adresse
[unteres Byte]
DIN
DOUT
D8...D15
DØ...D7
dyn. Speicherbank

Bild 4.36

Ausgängen gleichzeitig an alle Speicherbausteine. Die R/$\overline{W}$-Leitung wird über einen Treiber direkt auf die $\overline{WE}$-Eingänge aller Speicher geführt. Der nicht invertierende Treiber muß entweder ein AmZ 8166 sein, oder es ist ein Widerstand von 33 Ω in die Zuleitung zu schalten. Man sollte nach dem Beispiel der $\overline{CAS}$-Erzeugung verfahren. R/$\overline{W}$ wird auf 4 Eingänge des 8166 gegeben und von 4 Ausgängen auf die Schreibeingänge der Speicher verteilt.

Bild 4.37 zeigt den Lesezyklus des Z 8000 mit dem Speicherwerk nach Bild 4.36. An der Lo ↑ Hi-Flanke von $\overline{AS}$ wird die Zeilenadresse übernommen. Die entsprechende RAS-Leitung ist aus der Dekodierung A0 und A15 abzuleiten. Die Adressen werden bekanntlich zu dieser $\overline{AS}$-Flanke gültig. Dieser Zusammenhang wurde in Abschnitt 2.8 ausgiebig dargestellt. Daraus folgt, daß die Zeilenadresse an der Hi ↓ Lo-Flanke von $\overline{RAS}$ übernommen wird. Die Zeilenadresse liegt an

Speicher lesen mit dem
Baustein 3242 (MUX und Auffrisch-Zähler)

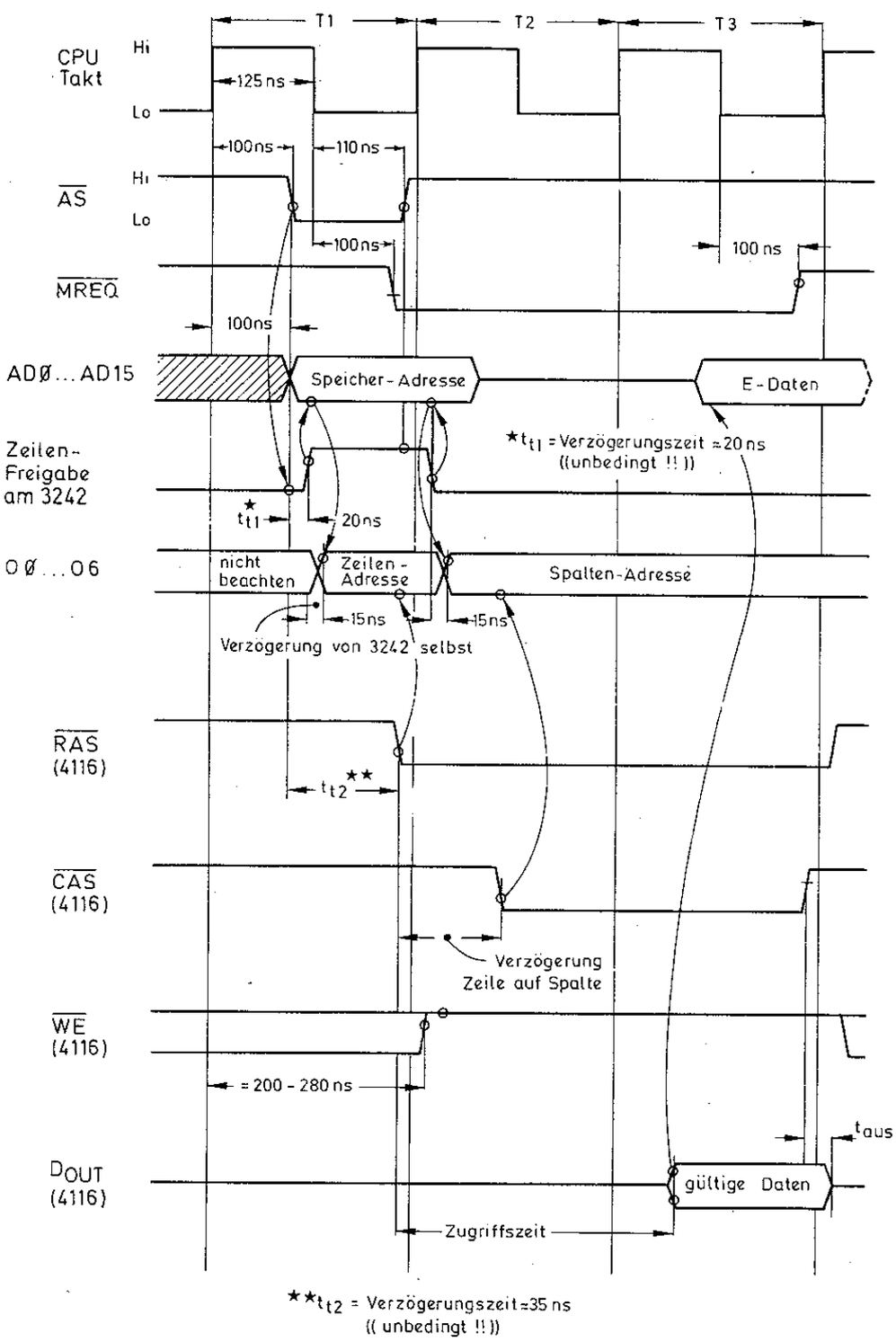


Bild 4.37

den Speichern an, weil der Zeilenfreigabeeingang am 3242 auf Hi liegt. Der Zeilenfreigabeeingang ist nach der angegebenen Verzögerung von 50 ns Lo, wenn eine RAS-Leitung aktiviert wurde. Man muß den vorhergehenden CPU-Lese- bzw. Schreibzyklus betrachten, um zu verstehen, wann der Zeilenfreigabeeingang Hi wurde. Das Signal $\overline{MREQ}$ wurde 80 ns nach der Hi↓ Lo-Flanke des dritten Maschinenzyklustakts Hi, sodaß nach allen Verzögerungen, die im System auftreten, der Zeilenfreigabeeingang etwa zur Hi↑ Lo-Flanke des ersten laufenden Maschinenzyklus von Lo auf Hi springt und somit die Zeilenadresse an den 3242-Ausgängen $\overline{O0}$ . . . $\overline{O6}$ wirklich anliegt. In Bild 4.37 ist der extremste Fall eingezeichnet, nachdem die **Zeilenfreigabe unbedingt Hi sein muß.** Ist das nicht der Fall, wird durch $\overline{RAS}$ an den Speichern eine ungültige Zeilenadresse übernommen. Auf diesen Sachverhalt ist also unbedingte Aufmerksamkeit zu richten,

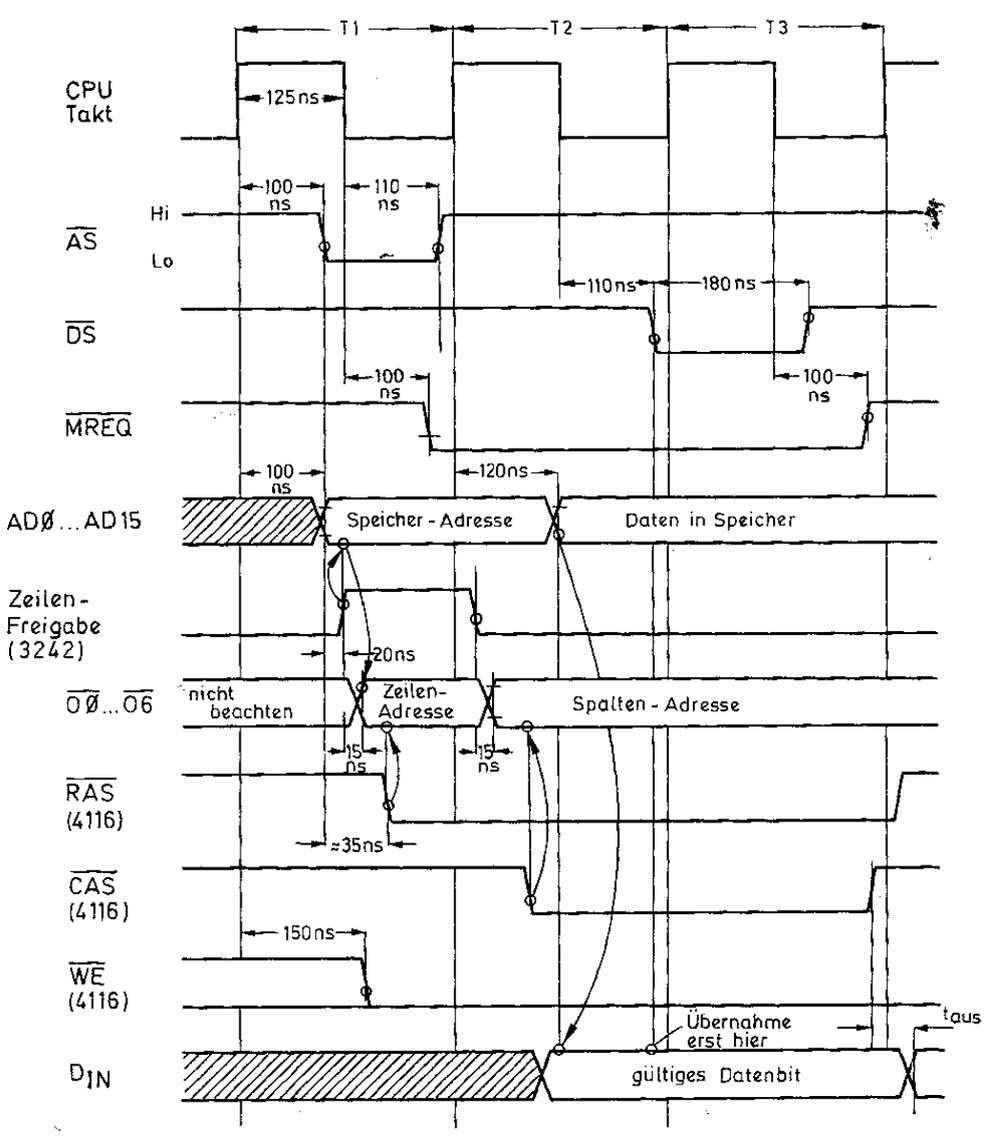


Bild 4.38

sonst hat man großen Ärger mit dem Speicherwerk. Dieses Fehlverhalten tritt in der Speicherschaltung auch nicht auf, wenn die Verzögerung von $\overline{RAS}$ auf die Zeilenfreigabe richtig dimensioniert wird (Verzögerung 40 . . . 50 ns, max. 50 ns). Letztlich ist das entscheidend wichtige Signal die Lo ↑ Hi-Flanke von $\overline{AS}$, auf der die ganze Schaltung funktioniert.

In **Bild 4.38** zeigt das Signaldiagramm für das Schreiben praktisch den selben Sachverhalt in grün. Wenn man das Lesen verstanden hat, wird Schreiben ebenso verstanden. In **Bild 4.39** ist in einem weiteren Diagramm auf die Erzeugung von $\overline{CAS}$ aus $\overline{DS}$ der CPU Bezug genommen. Man kann $\overline{DS}$ deshalb zu diesem Zweck verwenden, weil die schnellen

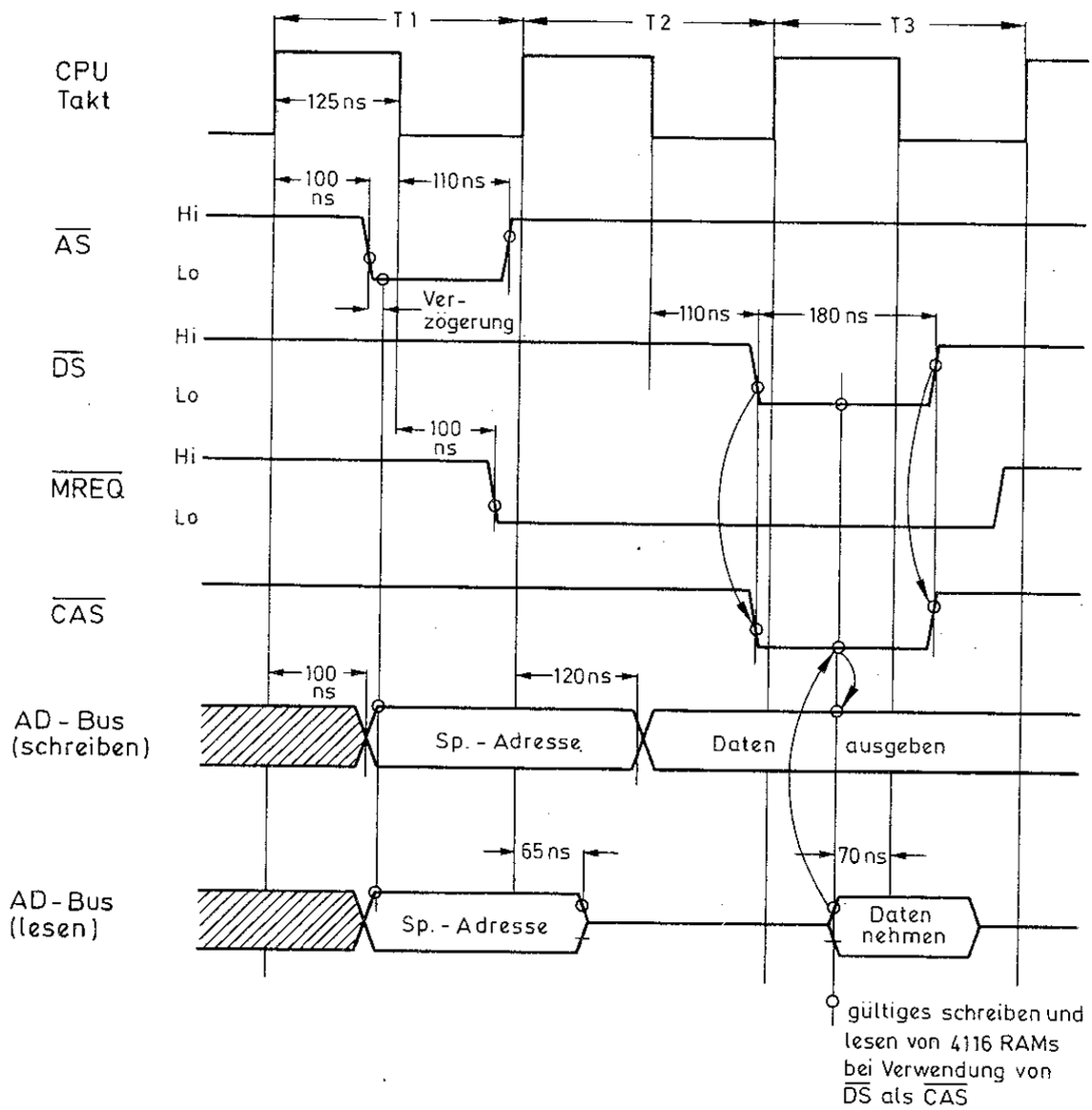


Bild 4.39

dynamischen Speicher schnell genug sind, innerhalb des aktiven $\overline{DS}$-Zeitraums die Daten zu liefern bzw. zu beschreiben sind.

Es gibt noch eine weitere Lösung, das Interface für dynamische RAMs zu vereinfachen. AMD bietet einen Speichersteuerbaustein an, der mit allen Logikelementen wie Adressen multiplexen, Zeilenfreigabe dekodieren und Refreshadresse erzeugen, ausgestattet ist. Der Baustein hat die Bezeichnung AmZ 8164 und ist in einem 40poligen DIL-Gehäuse untergebracht (Anschlußbezeichnung in **Bild 4.40**). Der Baustein besitzt die Möglichkeit, eine Adresse über einen Maschinenzyklus zu speichern, das an dem $\overline{EN}$-Eingang (Pin 36) durch eine Lo ↑ Hi-Flanke ausgelöst wird. Die 16 Adreßeingänge können somit direkt von der CPU angesteuert werden. **Bild 4.41** zeigt den Zusammenhang. 16 Adreßeingänge (gegenüber 14 beim 3242) bieten eine weitere, flexible Anwendung von Speicherbausteinen unterschiedlichen Formats. Es lassen sich folglich aus 8 Zeilen- und Spaltenadreßbits auch 64K x 1 dynamische RAM-Bausteine einsetzen. Die Adreßleitung A15 trifft intern die Auswahl für den Refreshzähler zur Ausgabe von 128 oder 256

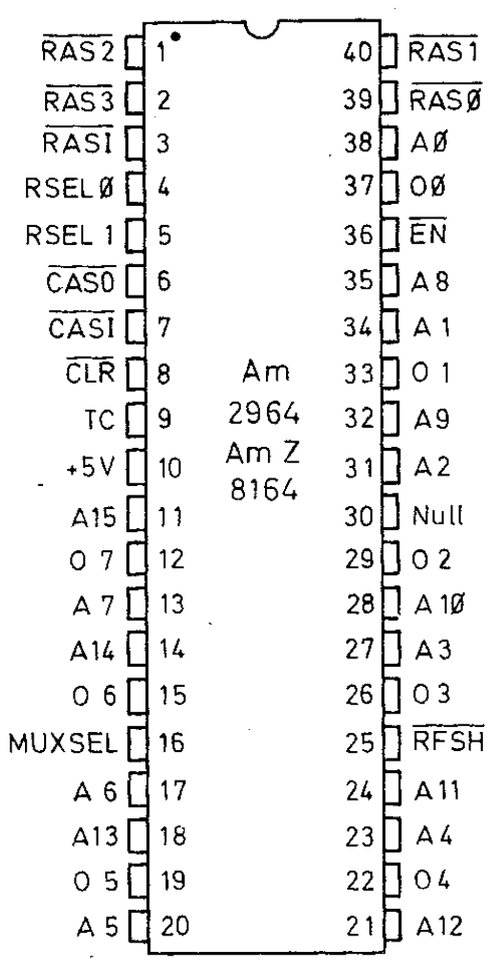


Bild 4.40

Zeilenadreßwerte entsprechend für 16K- oder 64K-RAM-Bausteine. Wird A15 nicht von einer CPU-Adresse belegt, ist der Eingang mit einem Widerstand auf Hi zu setzen (nach Angaben auf +12 V). Die Zeilendekodierung wird mit einem internen 4 aus 2 Dekoder gebildet, der vier Lo-aktive Zeilenfreigabesignale $\overline{RAS\,0}$ . . . $\overline{RAS\,3}$ erzeugt. Die beiden Adreßleitungen, die für die Zeilendekodierung herangezogen werden, sind bei $\overline{EN}$ gleich Hi ebenfalls mit den allgemeinen Adressen zwischengespeichert. Der Dekodierer hat den Bedingungseingang $\overline{RASIN}$ an Pin 3, der für eine Aktivierung einer RAS-Leitung erfüllt sein muß, z.B. $\overline{MREQ}$ muß gleich Lo sein. Der Baustein AmZ 8164 hilft, ein Interface zum Betrieb dynamischer RAMssmit der Z 8000-CPU einen weiteren Grad zu vereinfachen und die Bausteinzahl zu verringern.

In **Bild 4.42** ist eine Schaltung eines 64 KB-Speicherwerks dargestellt, die den Speichersteuerbaustein AmZ 8164 zum Einsatz bringt. Auf den ersten Blick ergeben sich Ähnlichkeiten zu der ersten Schaltung in Bild 4.36. Dieser Eindruck wird auch bei genauerer Betrachtung nicht abgeschwächt. Die Adreßeingänge A0 . . . A15 des 8164 werden direkt oder aus einem getrennten Adreß-/Datenbus versorgt, wobei $\overline{EN}$ (Zwischenspeicherfreigabe) mit $\overline{AS}$ angesteuert wird oder bei getrenntem Bus auf Null gelegt wird, — je nach Art des Adreßbusses. Der Steuereingang MUXSEL schaltet bei einer Hi↓Lo-Flanke von der Zeilenadresse A0 . . . A7 (MUXSEL gleich Hi) auf die Spaltenadresse A8 . . . A15 (MUXSEL gleich Lo) um. Es sei hier angemerkt, daß bei der Verwendung von 16K-RAMs der Eingang A7 selbstverständlich nicht benutzt werden darf;

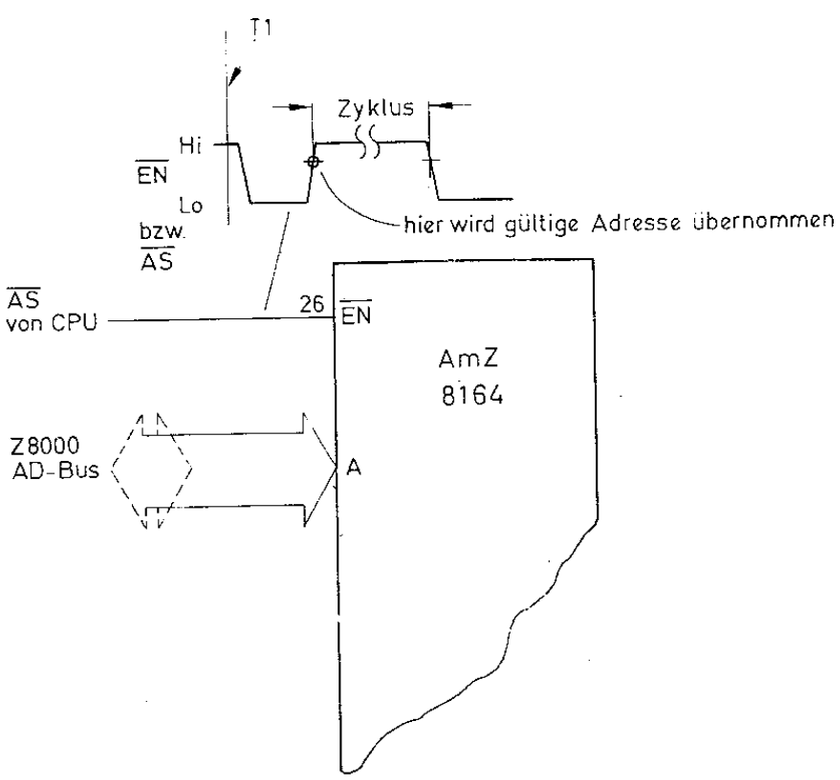


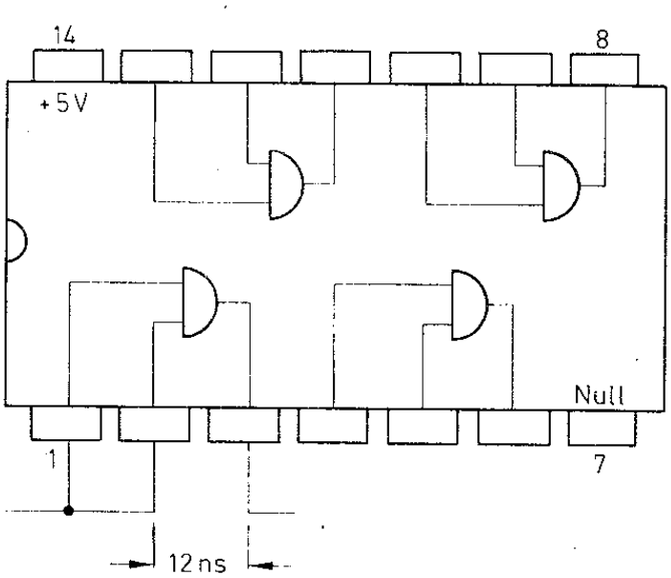
Bild 4.41

AS
von CPU (kann)
EN
8164
AmZ
Adressen von CPU
für 64K dyn. RAMs
CPU-Status Nr.8 = MEMSEL
MREQ
CASIN
RASIN
MUXSEL
CLR
RFSH
CASOUT
RAS3
RAS2
RAS1
RAS0
Adreß-Treiber
AmZ 8166
RAS/CAS-Treiber
8166
CAS/RAS
Verzögerung
max.37ns
LS21
DS
R/W
33Ω
A7 bleibt unbelegt
siehe Text und Bild
DATEN
D0 . . . D7
D8 . . . D15
CAS u
CAS o
RAS u
RAS o
WE
DIN
DOUT
8bit
8144
OE
0 . . .16KB
dyn. RAM
4116
(Am 9016)
17. . .32 KB
ungerade Adresse
[unteres Byte]
gerade Adresse
[oberes Byte]

Bild 4.42

ebenso auch A15 nicht, was jedoch als klar erscheinen muß. Für die 16K-RAMs reicht die Zeilenadresse nur von A0 . . . A6. Würde man A7 am 8164 mit einer Speicheradresse belegen, würde die Spaltenadresse für das Speicherwerk grundsätzlich falsch (um eine Stelle verschoben) ausgegeben werden. Der Eingang MUXSEL wird von dem Vierfach-UND-Gatter mit folgender Verzögerung angesteuert. Ist kein RAS-Signal aktiviert (Lo), wird an MUXSEL demzufolge Hi anliegen. Im Ruhezustand wird also immer die Zeilenadresse an den Multiplexausgängen O0 . . . O7 anliegen ($\overline{O7}$ wird nicht benutzt).

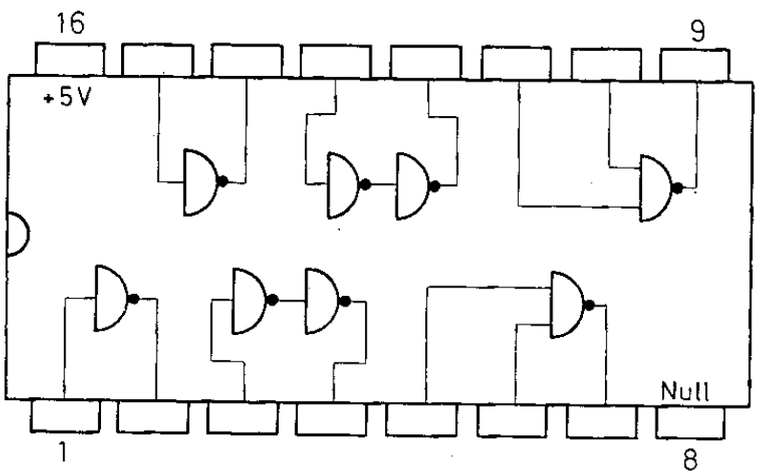
Auch in dieser Schaltung tritt das gleiche Problem mit der Verzögerung wie in der Schaltung 4.36 auf, das die Umschaltung an MUXSEL betrifft. War der vorhergehende Zyklus ebenfalls ein Speicherzyklus, wird erst nachdem $\overline{MREQ}$ gleich Hi wurde, auch eine der RAS-Leitungen wieder deaktiviert, wonach MUXSEL nach der Verzögerung wieder Hi wird. Für die Verzögerung ist 40 . . . 50 ns (max. 50 ns) anzusetzen, wobei die Verzögerung als Ganzes zu sehen ist. Die Verzögerung an sich muß zu der Signallaufzeit des UND-Gatters (Verzögerung 12 ns) als Summesgesehen werden, d.h. die Verzögerung darf 37 ns nicht überschreiten. Diese kleine Rechnung gilt auch für die Verzögerung in der Schaltung nach Bild 4.36. Der Steuerbaustein AmZ 8164 bietet jedoch gegenüber der Schaltung mit dem 3242 eine allgemein geringere Verzögerung, so daß der kritische Fall einer falschen Adreßübernahme noch mit geringerer Wahrscheinlichkeit auftritt, als dies bei der ersten Schaltung ohnehin schon mit gering zu bezeichnen ist. Als Beispiel für eine Dimensionierung der Verzögerung mit realen Gattern ist in **Bild 4.43** eine Auswahl an verschiedenen Typen dargestellt, die für beide Speicherwerke (Bild 4.42 und 4.36) verwendet werden können.



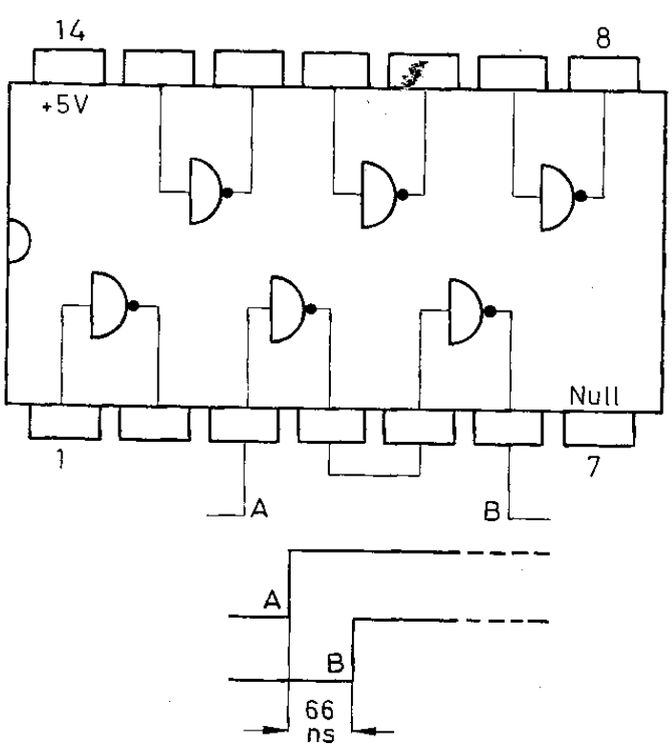
74 LS 08 4 UND Verzögerung typ. 12 ns

Bild 4.43

Signalverzögerung mit digitalen
(nicht-) invertierenden Gattern



FLH 641 (49703)
mittlere Laufzeit: typ. 10 ns, max. 18 ns pro Gatter



74L04 6 Inverter mit einer Impulsverzögerung
von 33 ns (typ. je Gatter)

Bild 4.43

Als Adreß- und $\overline{RAS}/\overline{CAS}$-Treiber wurden wieder die spezifischen Typen AmZ 8166 (nicht invertierend) verwendet, von dessen Ausgängen die jeweiligen Speichereingänge angesteuert werden. Der $\overline{RFSH}$-Eingang kann statisch auf Hi gelegt werden, weil für den Z 8000 der Zeilenadreßzähler im 8164 nicht benötigt wird. Für den Prozessor 8085 ist diese Funktion bei einem dynamischen Speicherbetrieb notwendig. Der Datenbus des Speicherwerks ist mit den Treiberbausteinen AmZ 8144 (nicht invertierende Gatter mit Tri-State-Ausgängen) an den allgemeinen Systemdatenbus anzukoppeln. Wenn man korrekt in Bytegruppen sowohl Ein- als auch Ausgänge der Speicher mit diesen Gattern treibt, sind insgesamt 8 Bausteine erforderlich. Die 16K-RAM-Bausteine haben getrennte Ein- und Ausgänge; dem muß man Rechnung tragen. In **Bild 4.44** ist das

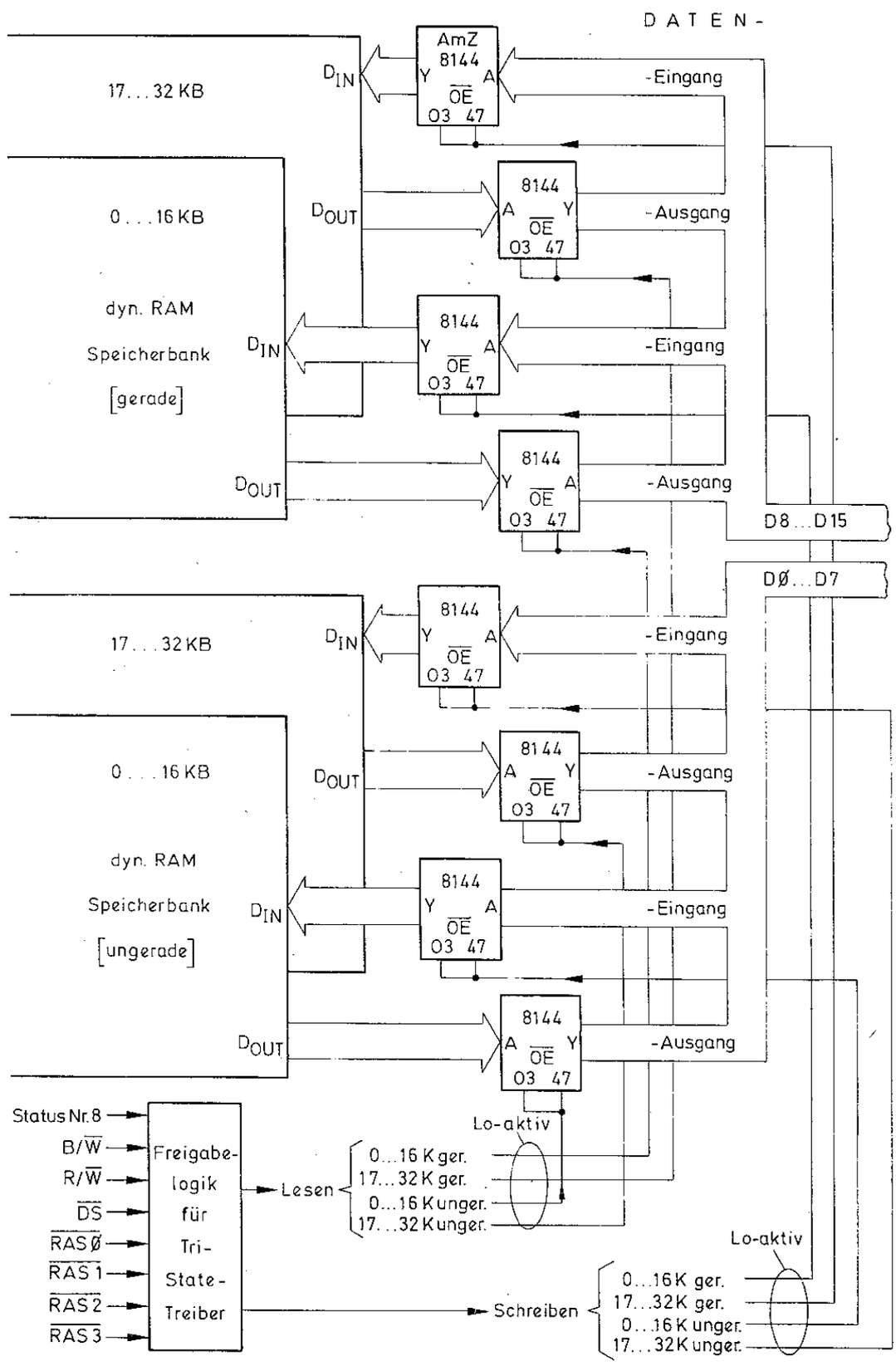


Bild 4.44

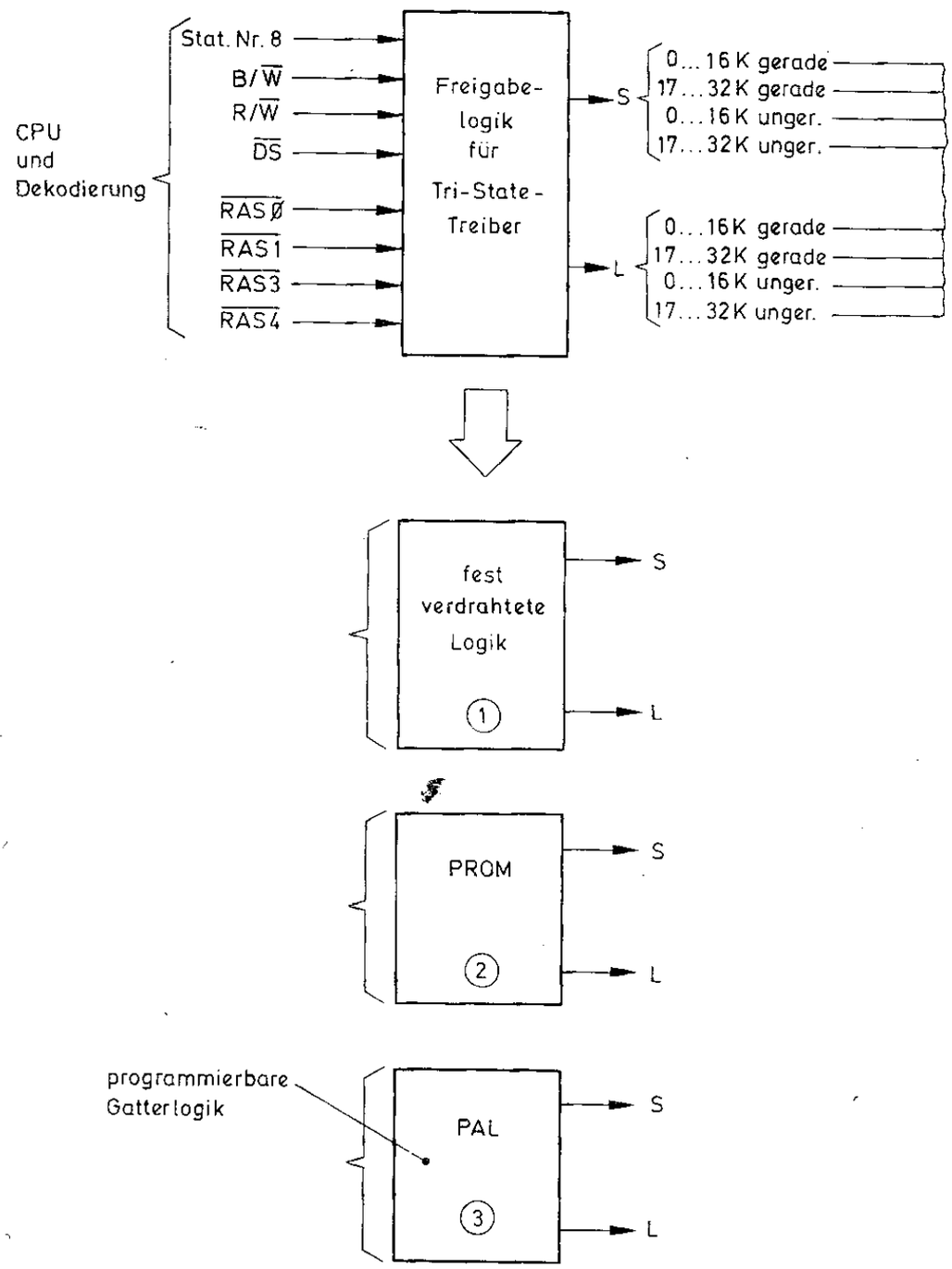

CPU
und
Dekodierung
Stat. Nr. 8
B/W
R/W
DS
RAS Ø
RAS 1
RAS 3
RAS 4
Freigabe-
logik
für
Tri-State-
Treiber
S
L
0...16 K gerade
17...32 K gerade
0...16 K unger.
17...32 K unger.
0...16 K gerade
17...32 K gerade
0...16 K unger.
17...32 K unger.
fest
verdrahtete
Logik
1
S
L
PROM
2
S
L
programmierbare
Gatterlogik
PAL
3
S
L

Bild 4.45

Datenbus-Interface der Schaltung nach Bild 4.42 noch genauer dargestellt. Jede Freigabeleitung $\overline{OE}$ eines Treiberbausteins wird in zwei Gruppen geordnet, die mit "Freigabe lesen" und "Freigabe schreiben" bezeichnet sind. Jeweils vier Leitungen erfordern die Ansteuerung aus einer Freigabelogik, die für diesen Fall geeignet entworfen werden muß. Das Datenbus-Interface ist in jedem Fall von Bedeutung, weil es für die Schaltung in Bild 4.36 genauso einzusetzen ist. Wie aus Bild 4.44 zu ersehen ist, sind insgesamt 8 Bedingungen für die Freigabe eines Treiberbausteins zu erfüllen. Sie müssen so verknüpft sein, daß normal ein Treiber aktiv ist. Maximal können es auch zwei Bausteine sein. Die RAS-Leitungen bestimmen, welche der Speichergruppen zum Transfer anstehen; aus $R/\overline{W}$ wird die Richtung bestimmt; $\overline{DS}$ gibt Aufschluß über den genauen Datentransfer-Zeitraum und $B/\overline{W}$ gibt an, ob ein oder zwei 8144 geöffnet werden (folglich aus Byte- oder Wortzugriff). Zur Realisierung einer Freigabelogik nach diesem Bedingungsmuster gibt es drei Möglichkeiten. **Bild 4.45** zeigt, welche Möglichkeiten offen stehen.

(1) festverdrahtete Logik
(2) Festwertspeicher, Format 256 x 8 Bit
(3) PAL-Baustein (programmierbare Gatterlogik)

Aus Bild 4.44 ist zu ersehen, daß die Logik 8 Ausgänge haben muß, 4 für lesen und 4 für schreiben. Die Anzahl der 8 Bedingungen legt fest, daß bei Verwendung eines PROMs die Kapazität 256 x 8 für die 8 Variablen erforderlich sind. Im folgenden werden einige Permutationen zusammengestellt, die angeben, welche Treiberbausteine angewählt werden. Die Ansteuerung der Freigabeleitung eines Treiberbausteins ist eine Funktion der jeweiligen Variablen, die links geschrieben sind:

				$\overline{RAS}$					
Stat. Nr. 8	**$B/\overline{W}$**	**$R/\overline{W}$**	**$\overline{DS}$**	**0**	**1**	**2**	**3**		**Treiber**
0	1	1	0	0	0	1	0	lesen	17 . . . 32 K gerade
0	0	1	0	0	0	1	1	lesen	17 . . . 32 K gerade 17 . . . 32 K ungerade
0	1	0	0	1	0	0	0	schreiben	0 . . . 16 K gerade
0	1	0	0	0	1	0	0	schreiben	0 . . . 16 K ungerade
0	0	0	0	1	1	0	0	schreiben	0 . . . 16 K gerade 0 . . . 16 K ungerade

(für alle RAS-Leitungen gilt jede 1 als aktiv Lo)

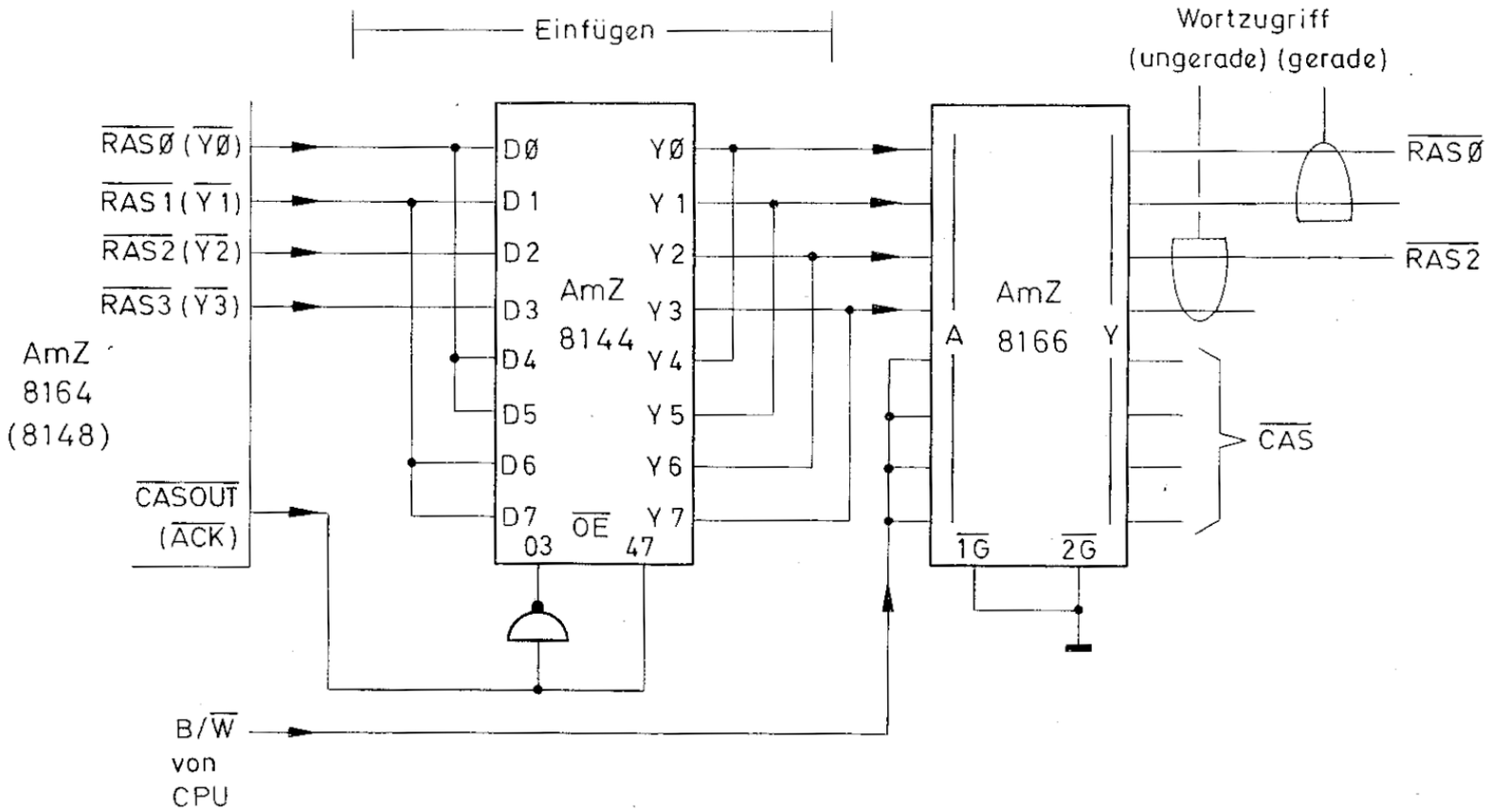
Die fünf aufgezeigten Permutationen zeigen in drei Fällen den Byte-Modus und in zwei Fällen den Wort-Modus, bei dem immer eine gerade und ungerade Speicherfeldgruppe geöffnet wird. Diese Tabelle läßt sich noch fortsetzen. Die Treiberbezeichnung sagt aus, daß an den jeweiligen $\overline{OE}$-Eingängen ein Losanliegen muß. Im Wort-Modus ($B/\overline{W}$ gleich Lo) werden zwei Treiberbausteine angewählt. Nach der komplett ausgearbeiteten Bedingungstabelle wird das PROM erstellt. Zwei Variable sind eigentlich Konstante. $\overline{DS}$ muß immer Lo sein und der Status Nr. 8 immer Lo oder Hi (aktiv nach Polarisierung des Statusdekoders). Für das gegebene Beispiel ist die Programmierung eines PAL-Bausteins sicher als vorteilhaft zu sehen, weil es sehr einfach geht. Man kann den gegebenen Fall beispielsweise mit dem Typ PAL 10L8 lösen. Es hat 10 Variableneingänge und 8 Gatterausgänge und ist Lo aktiv. Diese interessanten Bausteine sind keine PROMs, wenn man sie auch programmieren muß, und andererseits

keine festverdrahtete Logik, wenn sie sich nach der Programmierung auch danach verhalten. Das einfache daran ist die Formulierung der Bedingungen, die in Form von logischen Pegeln vom Eingang zum Ausgang nur zugeordnet werden müssen. Mit Hilfe eines PAL-Assemblers werden diese Zuordnungen in Programmieranweisungen umgesetzt. Der programmierte PAL-Baustein verhält sich dann genau nach den Bedingungen, die in der obigen Tabelle enthalten sind. Es erübrigt sich, heute an einer festverdrahteten Logik herum zu knobeln. Diese Zeiten sollten der Vergangenheit angehören. Bei Einsatz eines PROMs muß die gesamte Liste der 256 Bytes aufgesetzt werden, die anschließend nach den bekannten Verfahren programmiert wird. Die Programmierung eines PAL ist sehr effektiv zu vollziehen. PAL-Bausteine werden von der Firma Monolithic Memories hergestellt. Für alle drei Methoden nach Bild 4.45 ist die Zeitbedingung für die Erzeugung einer Freigabe mit etwa 40 ns anzugeben. Ein PAL-Baustein des Typs 10L8 ist mit einer Signallaufzeit von 25 ns angegeben (typisch von einem beliebigen Eingang zu einem beliebigen Ausgang).

Eine wichtige Hardwarekomponente in den Schaltungen nach Bild 4.42 wie auch Bild 4.36 muß noch angefügt werden. Dieses Detail kommt auch in der zuvor genannten Tabelle zum Ausdruck. Beide Speicherwerke müssen die Möglichkeit eines wahlfreien Zugriffs im Byte- und Wortmodus enthalten, das die CPU an ihrem Ausgang B/$\overline{W}$ anzeigt. Generell sind die gezeigten Speicherwerke nur für den Bytezugriff vorbereitet, d.h. nur wenn B/$\overline{W}$ Hi anzeigt, wird ein Transfer erfolgen. Bei B/$\overline{W}$ gleich L funktioniert die Schaltung nicht, weil immer nur eine 16 KB-Gruppe aktiviert wird. Diesen Fall kann man mit einer kleinen Änderung in beiden Speicherwerken leicht korrigieren, so daß Byte- und Wortmodus möglich werden. Die Schaltung in **Bild 4.46** zeigt die Änderung. Es wird einfach ein Treiberbaustein AmZ 8144 zwischen RAS-Dekodierausgängen und Leitungstreiber AmZ 8166 eingefügt. Das Signal B/$\overline{W}$ steuert die getrennten Freigaben $\overline{OE}$ 03 invertiert und $\overline{OE}$ 47 nicht invertiert an. Beim Bytezugriff werden von allen 4 RAS-Leitungen bekanntlich nur eine aktiv. Beim Wortzugriff werden jeweils $\overline{RAS\ 0}$ und $\overline{RAS\ 1}$ (oberes und unteres Byte des Bereichs 0 . . . 16K) und $\overline{RAS\ 2}$ mit $\overline{RAS\ 3}$ (oberes und unteres Byte des Bereichs 17 . . . 32K) aktiv. Da $\overline{CAS}$ generell für alle 4 Speichergruppen aktiv wird, ist mit der Verknüpfung der RAS-Leitungen am Eingang des AmZ 8144 der Wortzugriff gesichert. In der Schaltung nach Bild 4.36 wird der Treiberbaustein AmZ 8144 am Ausgang des Dekodierers AmZ 8148 eingefügt, an dem der Bedingungseingang E3 auf Hi gelegt werden muß.

Für das Speicherwerk in Bild 4.42 wird noch in Ergänzung zu den Signaldiagrammen in Bild 4.37, 4.38 und 4.39 der Zeitunterschied bei der Aktivierung von $\overline{DS}$ beim Lesen und Schreiben dargestellt. Aus dem Signal $\overline{DS}$ der CPU wird die Spaltenadreßübernahmes $\overline{CAS}$ bei den dynamischen RAMs abgeleitet. **Bild 4.47** zeigt den Unterschied zwischen lesen und schreiben bei der Aktivierung von $\overline{DS}$.

In **Bild 4.48** wird nochmals an einen eventuell einzuführenden Wartezyklus erinnert. Der WAIT-Eingang der CPU muß während der Hi ↓ Lo-Flanke im zweiten Maschinenzyklus gleich Lo sein (mindestens 70 ns vor der Flanke). In **Bild 4.49** ist der Signalverlauf dargestellt, wie er am Steuerbaustein AmZ 8164 bezüglich des Eingangs MUXSEL und der Aktivierung von $\overline{CAS}$ abläuft. Ist MUXSEL gleich Lo geworden, darf $\overline{CAS}$ nicht unmittelbar gleichzeitig Lo werden, weil dies gerade in den Umschaltzeitraum von der Zeilen- auf die Spaltenadresse fällt, in dem die Adresse an $\overline{00}$ . . . $\overline{07}$ als nicht stabil gilt. Man muß eine Verzögerung bis zum Zeitpunkt einer stabilen Adresse schon einhalten, bevor $\overline{CAS}$ aktiviert wird. Bereits nach 30 ns nach der Hi ↓ Lo-Flanke an MUXSEL (Umschaltung Zeilenadresse auf Spaltenadresse) kann die Spaltenadresse



beim 8148: Pin E3 von B/$\overline{W}$ abkoppeln und auf Hi legen.

Bild 4.46

Speicherzugriff

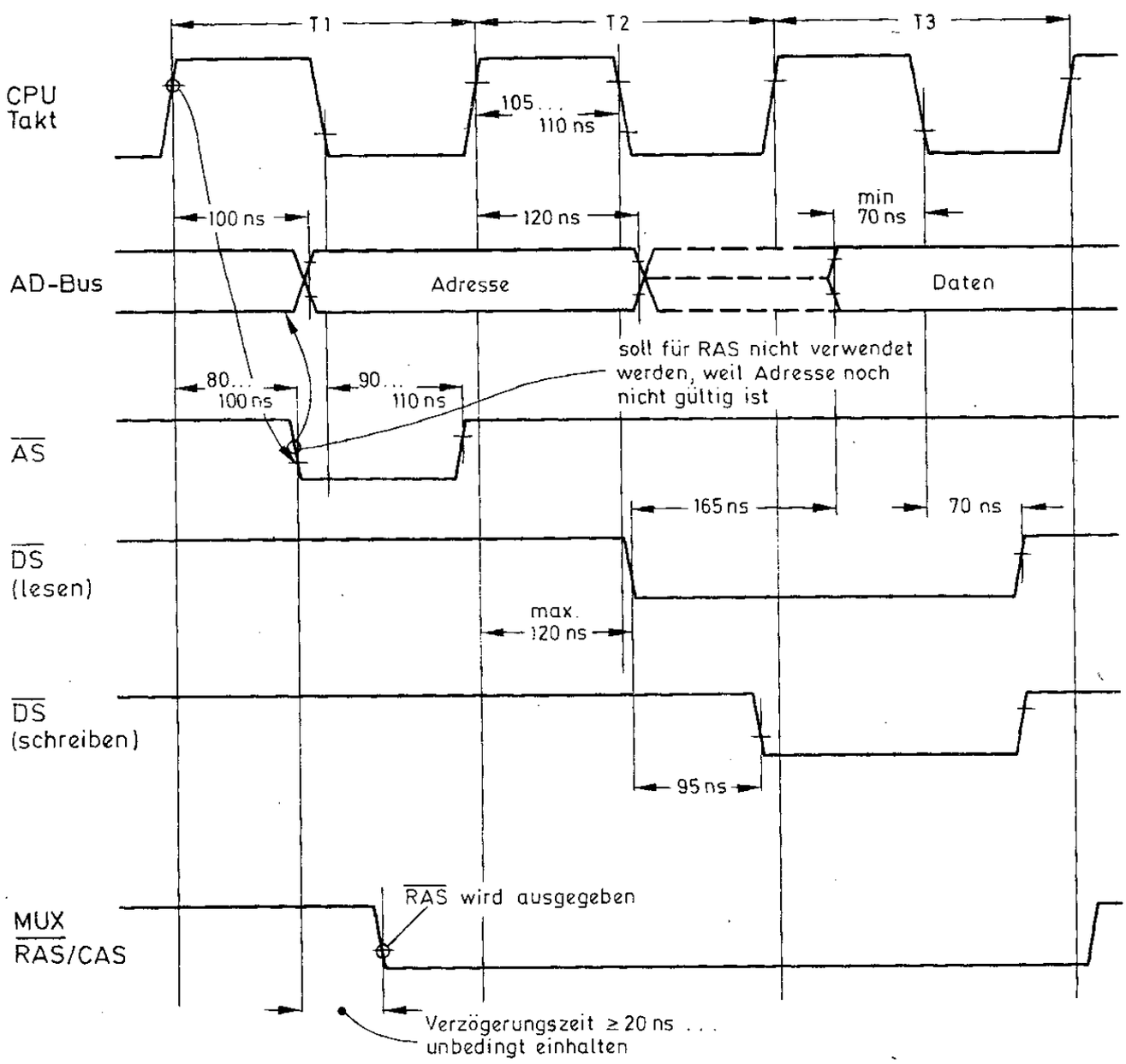


Bild 4.47

Speicherzugriff

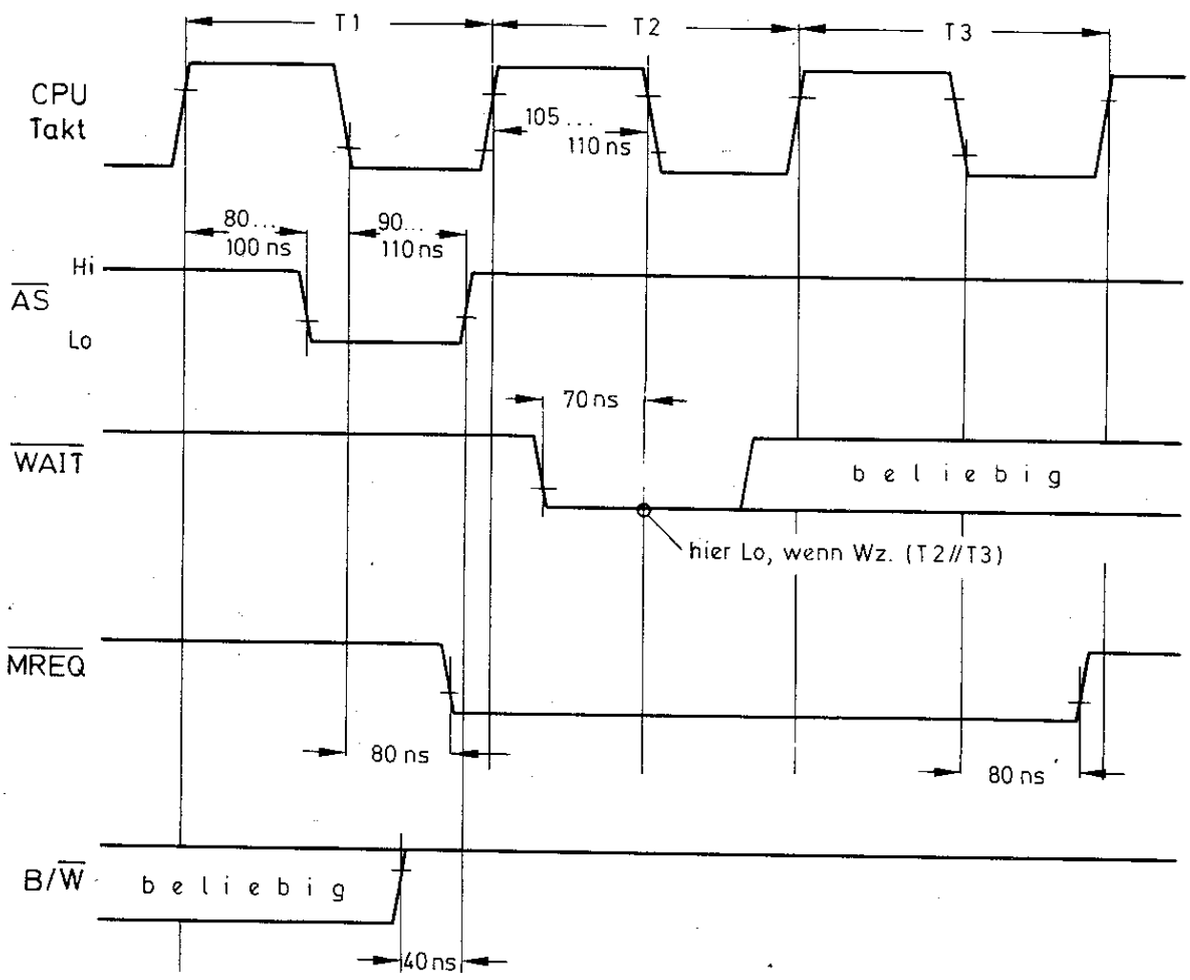


Bild 4.48

als gültig angesehen werden. Normal kommt der Fall einer so frühzeitigen Aktivierung von $\overline{DS}$ auch nicht vor, zumal zwischen $\overline{DS}$ aktiv und $\overline{CAS}$ aktiv eine weitere Verzögerung von einigen Nanosekunden liegt. In Bild 4.49 handelt es sich auch um einen extremen Fall. Würde $\overline{DS}$ zu früh aktiviert werden, kann es sich auch um einen Fehler in der CPU handeln. Das ist jedoch absolut selten. Man kann als gewisse Regel angeben, daß die Spaltenadresse etwa zur Hi ↓ Lo-Flanke im zweiten Maschinentakt übernommen wird. Ist der Einsatz von 16K-Typen mit 200 ns Zugriffszeit vorausgesetzt, kann ein Wartezyklus der CPU erforderlich sein; im allgemeinen jedoch nicht. Letztlich muß ein Speichertestprogramm entscheiden.

Von den 16K-RAM-Typen mit einer

Speicherzugriff

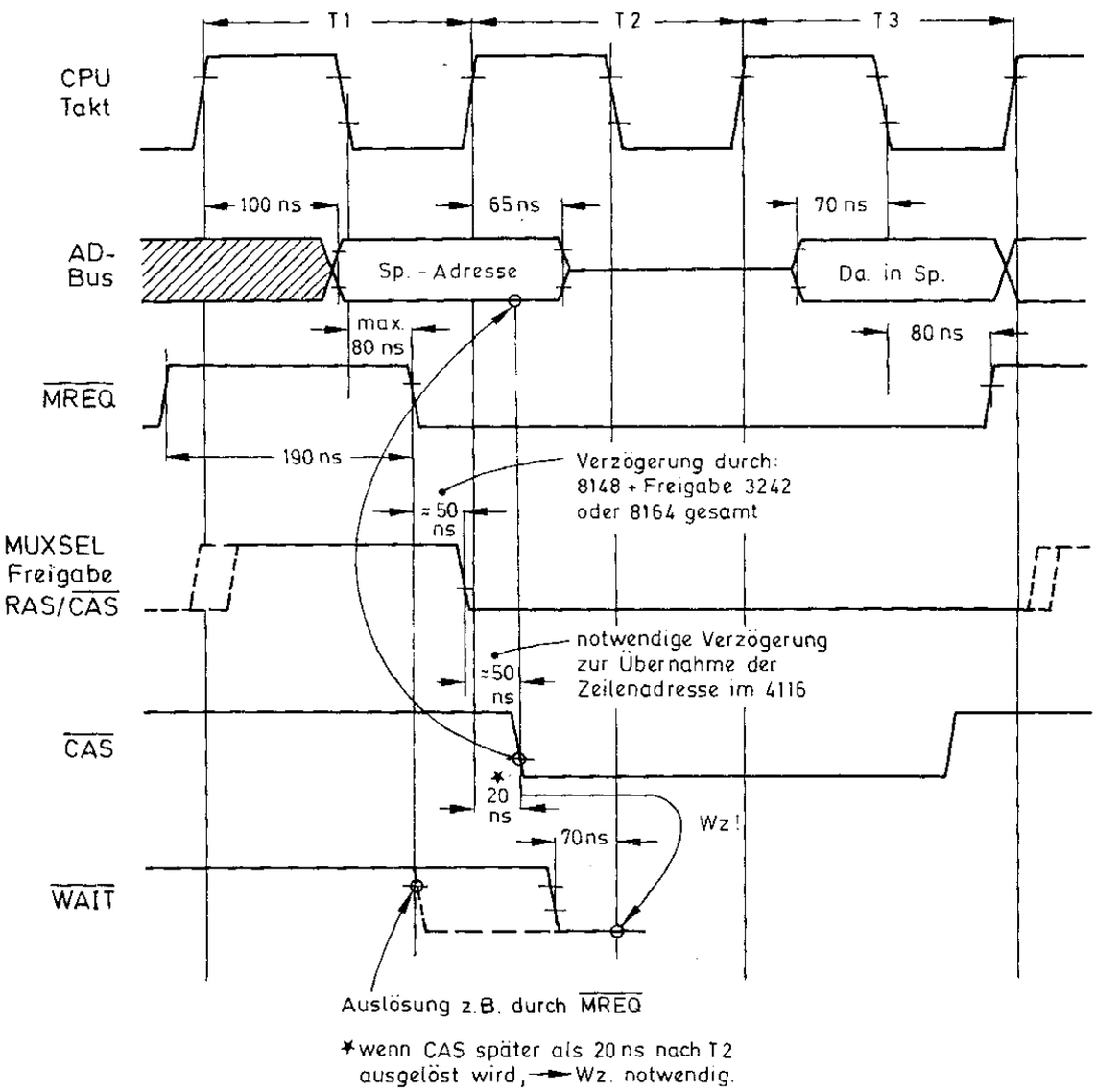


Bild 4.49

Versorgungsspannung von nur +5 V war bisher nur kurz die Rede. Der MK 4516 von Mostek ist einer dieser 5 V-Typen. Worin liegt ihr Vorteil? Er ist nicht allein in der einfachen Versorgung zu sehen. Das 16 K-Chip hat einen separaten Vorspannungsgenerator auf dem Substrat, der es auf etwa –3 V negativ einpegelt. Ein absoluter Fortschritt bei diesen Typen ist im Leistungsverbrauch während des aktiven Zustands zu notieren. Der MK 4516 benötigt nur 140 mW gegenüber dem MK 4116 mit 462 mW, d.h. der 5 V-Typ benötigt eine um über das Dreifache geringere Leistung. Ferner ist zu bemerken, daß die 5 V-Typen noch schneller wurden. Der MK 4516-8 ist mit dem extremen Wert von 80 ns Zugriffszeit angegeben. Der "langsamste" Typ MK 4516-12 erfüllt mit 120 ns Zugriffszeit immer noch die Bedingung, daß ein Z 8000A mit 6 MHz Taktfrequenz einen Transfer ohne Einschieben eines Wartezyklus ausführen kann. Ein weiterer Kommentar ist hierzu überflüssig. Diese dynamischen RAMs qualifizieren sich damit zu den künftigen Standard-Speicherbausteinen für größere Z 8000-Systeme.

Für den 5 V-Typ von Mostek ist eine zusätzliche Eigenschaft zu vermerken. **Bild 4.50** zeigt dies. Der zweite existierende 5 V-Typ mit der Bezeichnung 2118 der Firma Intel unterscheidet sich vom MK 4516 in einer Kleinigkeit. Der Mostek-Typ besitzt noch einen internen Auffrisch-Modus, der an Anschluß 1 aktiviert wird. Ein interner Zähler gibt

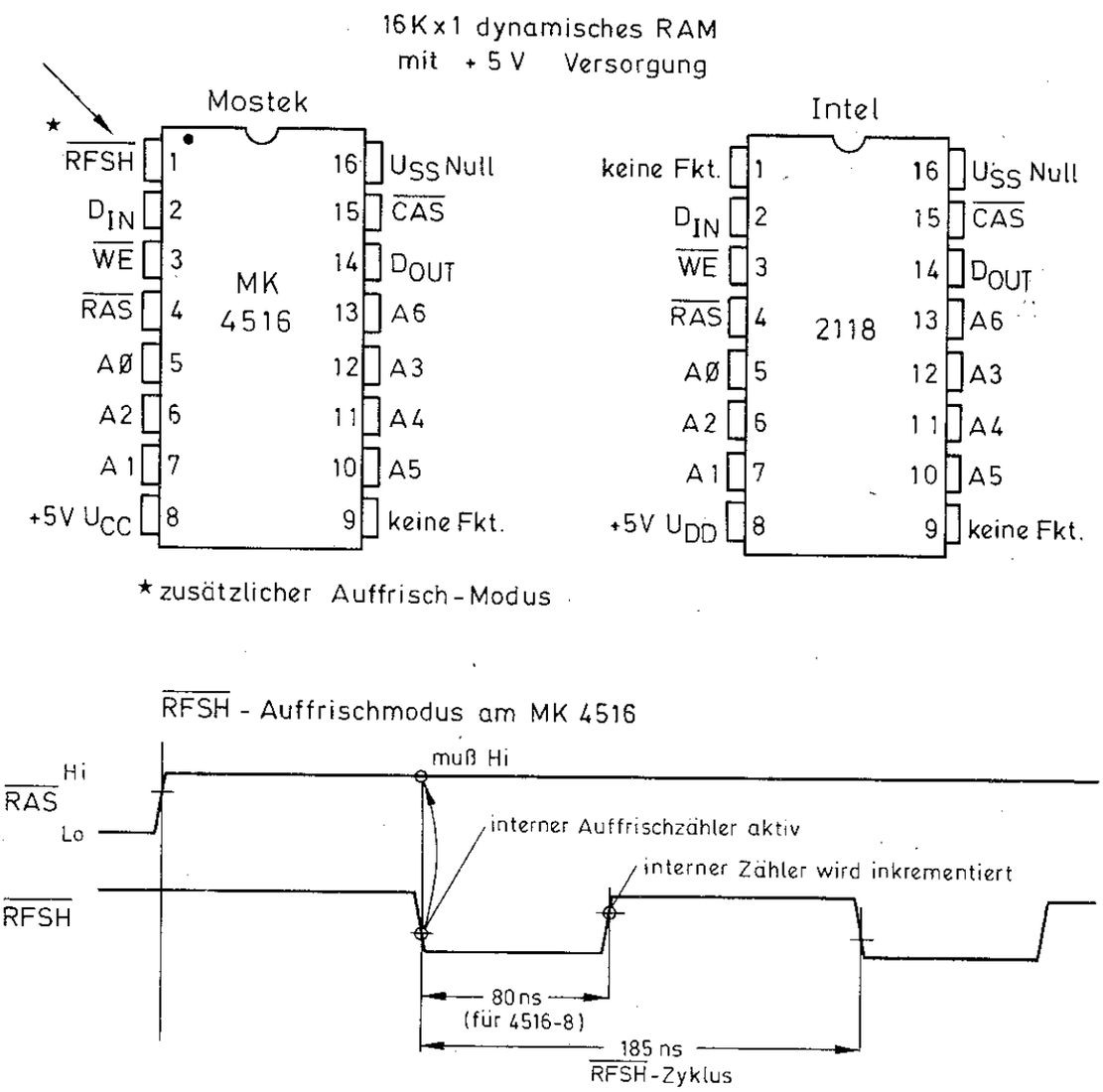


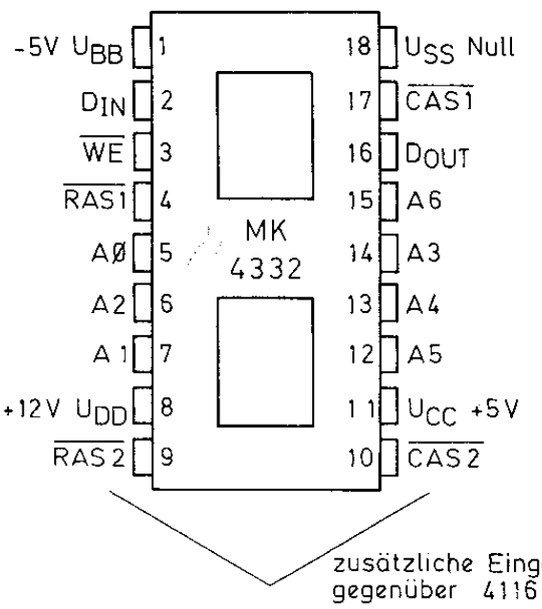
Bild 4.50

eine Zeilenadresse aus, die zum Auffrischen benutzt wird. Ein Zyklus umfaßt 185 ns, in dem alle Zellen einer Zeile aufgeladen werden. An der Lo ↑ Hi-Flanke am Eingang $\overline{RFSH}$ wird der Zähler inkrementiert und damit auf die nächste Zeile eingestellt. Beim Z 80000-Einsatz wird man wohl den Refresh CPU-seitig ausführen, obgleich ein Refreshmodus mit diesem Speicher gewisse Zeitersparnisse bringen kann. Der CPU-Zyklusrefresh dauert unweigerlich 750 ns (bzw. 500 ns bei der A-Version der CPU). Man muß es sich eben überlegen, was man tun will. Der CPU-Refreshzyklus beinhaltet keine Aktivierung des $\overline{DS}$-Signals, so daß an den Speicheradreßeingängen immer nur eine Zeilenadresse anliegen wird. Der Speicher interne Refresh funktioniert genauso, nur mit dem kleinen Unterschied, daß die interne Refreshadresse schneller am Bestimmungsort ist.

Nebenbei sei zu den 5 V-Typen noch bemerkt, daß der MK 4516 in seiner schnellsten Version den Intel-Typ um 20 ns in der Zugriffszeit unterbietet (Intel 2118-3 → 100 ns) und noch um 10 mW weniger Leistung verbraucht (Intel 2118 → 150 mW). Es sollte wirklich nur nebenbei bemerkt sein, denn die Differenzen sind nichts weiter als Nuancen.

4.4.3 Das 32K x 1 bit dynamische RAM

Man kann aus einem technischen Umstand eine Tugend machen. Die technische Gegebenheit bei dynamischen RAMs mit Multiplexadreßeingängen ist die, daß die effektive Zahl der Adreßleitungen (Zeilen- plus Spaltenleitungen) intern immer geradzahlig ist, d.h. 6 Anschlüsse ergeben 12 Leitungen (demzufolge 4K-Typ), 7 Anschlüsse des 4116 ergeben 14 Leitungen (16K-Typ) und 8 Anschlüsse ergeben 16 Leitungen für das 64K-RAM. Ein 32K dynamisches RAM paßt hier nicht in dieses Speichergefüge. Diese technische Gegebenheit der Multiplex-RAMs löst keinen Notstand aus, der ein tugendhaft schnelles Bestreben einer Korrektur bewirkt; es besteht lediglich ein Bestreben der Hersteller nach einem stetigeren Angebot der Speicherdichte dynamischer RAMs, das es gilt, ohne großen technischen Aufwand zu initiieren. Woher genau die Lösung eines Tandembausteins mit dem Format 32K x 1 bit entsprungen ist, kann nicht mit Bestimmtheit angegeben werden. Die LSI 11/23 beispielsweise hat auf dem 40poligen Gehäuse ihres Zentralbausteins ebenfalls zwei Chips aufgesetzt bekommen (CPU-ALU-Datenchip und Mikroprogramm-Steuerchip). Wer von wem inspiriert wurde, ist belanglos. Es gilt der Grundsatz: "Was funktioniert, ist auch erlaubt". Dieser Grundsatz ist in der Computertechnik einschränkungslos anwendbar, nicht zu verwechseln mit weiteren Lebensbereichen; das gäbe ein Chaos. Für den Computerbausteinbereich gilt jedoch, daß die Lösung, zwei 4116-Chips auf einen 18poligen Baustein zu kleben, völlig logisch und auch hinreichend ist, wenn es funktioniert. An einer Funktion ist kein Zweifel zu hegen. **Bild 4.51** zeigt dieses dynamische Tandem-RAM von Mostek mit der Bezeichnung MK 4332. Die Zeilen-/Spaltenadreßeingänge beider Chips sind parallel an die Anschlüsse gelegt; ebenso der Schreibeingang und die Datenein-/-ausgänge. Nur die Zeilen- und Spaltenübernahmeeingänge sind separiert. Es ist tatsächlich im Baustein nur verbunden, was äußerlich an zwei diskreten Bausteinen verbunden würde, wenn über 16 KB ausgebaut wird. Das Resultat steht zahlenmäßig mit 18 Anschlüssen gegen 32 Anschlüsse zweier 16K-Typen. Das ist Grund genug, der für eine Tandemlösung spricht.


-5V U_{BB}
1
D_{IN}
2
$\overline{WE}$
3
$\overline{RAS1}$
4
A Ø
5
A2
6
A1
7
+12V U_{DD}
8
$\overline{RAS2}$
9
18
U_{SS} Null
17
$\overline{CAS1}$
16
D_{OUT}
15
A6
14
A3
13
A4
12
A5
11
U_{CC} +5V
10
$\overline{CAS2}$
MK
4332
zusätzliche Eingänge
gegenüber 4116

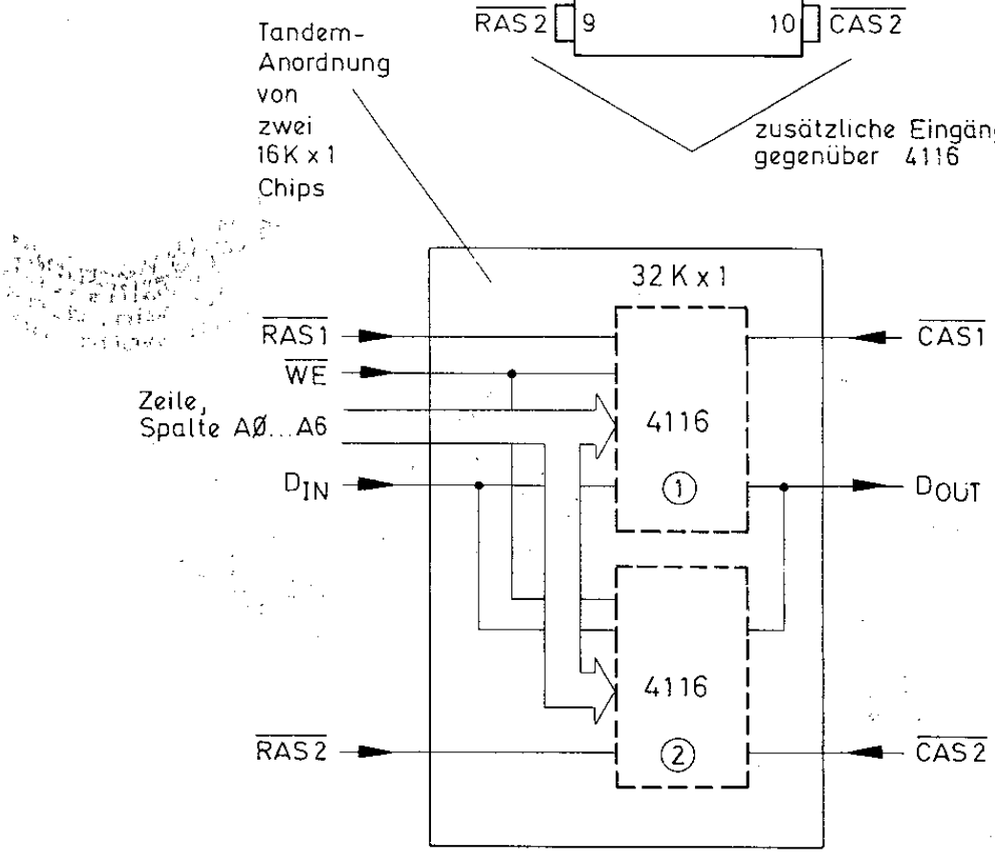

Tandem-
Anordnung
von
zwei
16K x 1
Chips
32 K x 1
$\overline{RAS1}$
$\overline{WE}$
Zeile,
Spalte AØ...A6
D_{IN}
4116
①
$\overline{CAS1}$
D_{OUT}
4116
②
$\overline{RAS2}$
$\overline{CAS2}$

Bild 4.51

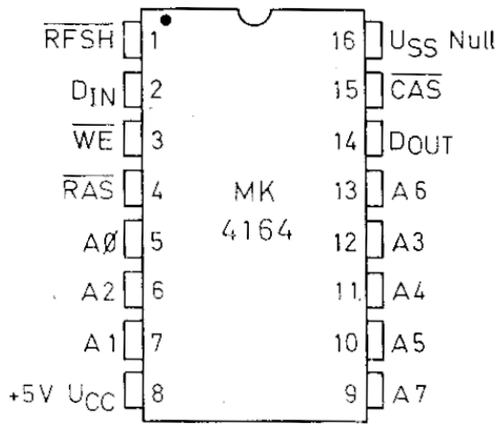
4.4.4 Beispiel einer 64K-Worte-Speicherbank mit 64K x 1 bit dynamischen RAMs

Der Bedeutung dynamischer RAM-Bausteine wird in Abschnitt 4.4 Rechnung getragen, indem das 64K x 1 bit-RAM in ein Speicherwerk mit eingeschlossen wird. Es gilt als absolutes Extrem im Halbleiterspeicherbereich und wird neben 5 V-16K-Typen der zentrale Schreib-/Lesespeichertyp der 80er Jahre sein. Mit 16 Bausteinen ist ein Segment des Z 8000 vollständig abzudecken, so daß der extensiven Adressierbarkeit des Z 8001 mit realer, adressierbarer Kapazität entgegnet werden kann. Vor allem ist mit einer maßvollen Quantität an Bausteinen einer absolut vielfältigen Anforderung benötigter Speicherkapazität nachzukommen. Ein Feld von 128K Worte benötigt 32 Bausteine, die für je ein Programmcode- und Datensegment mit 64K Worte dimensionierbar sind. Beide Felder passen mit kompletter Dekodierung ohne große Mühe auf eine Platine mit Doppel-Europaformat. Überschlägt man kurz den Energieaufwand für den Betrieb dieser Platine, hat man im aktiven Betrieb mit nicht mehr als etwa 7,5 W zu rechnen. Es ist schwer, einen geeigneten Kommentar zu finden, der auszudrücken vermag, wie hochklassig dieser Wert ist. Eines dürfte in jedem Fall klar sein, daß diverse Ölscheichs künftig am Energieverbrauch von Computern mit solchen Bausteinen nicht reich werden können. Im Rahmen der Weiterentwicklung von Solarzellen könnte ein mittleres Z 8000-System rein mit Sonnenenergie betrieben werden, wobei ein halber Quadratmeter Solarzellenfläche genügen kann.

Bild 4.52 zeigt das Anschlußbild des 64K x 1 bit-RAM des Typs MK 4164 von Mostek. Der Vergleich der Anschlußbezeichnungen zum 16K-Typ ergibt nur den einen signifikanten Unterschied, daß der +5 V-Anschluß von Pin 9 auf Pin 8 verlegt wurde. An Pin 9 des 4164 ist der höchstwertige Adreßeingang A7 angelegt. Weitere Spannungen sind nicht erforderlich. Signaltechnisch ist der 4164 zum 16K-Typ identisch. Der Vergleich zum 5 V-Typ 4516 zeigt noch die Übereinstimmung zu dem internen Refresh-Modus. Die Version MK 4164-10 ist die schnellste Ausführung mit einer Zugriffszeit von 100 ns. Die bereits erwähnte Reduzierung des Leistungsverbrauchs ist bemerkenswert. Der Baustein verbraucht im aktiven Zustand ($\overline{RAS}$ → Lo) 300 mW und im passiven Zustand 20 mW. Damit ist der Verbrauch noch um den Faktor 2 geringer als bei dem 5 V-Typ 4516, wenn man die Anzahl der Speicherzellen in Beziehung setzt. Bemerkenswert ist ferner, daß die interne Feldorganisation mit 128 Zeilen je Feld beibehalten wurde. Ein Feld hat die Organisation 128 Zeilen x 256 Spalten. Die Gesamtzahl der Spalten beträgt somit 512, die zahlenmäßig gleich der Anzahl der dynamischen Abfrageverstärker ist. Das Beibehalten der Zeilenzahl von 128 ermöglicht den Refresh aller Zellen in den typischen 2 ms.

Bild 4.53 zeigt die Schaltung eines Speicherwerks mit der Kapazität von 64K Worte. Es gibt für diesen Speichertyp keine andere Möglichkeit als das Wortformat. Das bedeutet, daß bei einem Bytezugriff der CPU für den Speicher trotzdem ein Wortzugriff erfolgt. Es besteht ernsthaft die Frage nach der Notwendigkeit, ob zwischen Byte und Wort der Aufwand zur Trennung lohnt, angesichts einer solch hohen Speicherdichte. Man darf jedoch nicht vergessen: Wird eine Adresse eines Speicherplatzes im Bytemodus beschrieben, bleibt das andere Byte des Speicherplatzes unbenutzt. Genaugenommen wird im Bytemodus das untere und obere Byte mit dem gleichen Wert geladen.

Der Bytetransfer ist insbesondere bei Befehlsholzyklen von Bedeutung. Beginnt die CPU mit der Ausgabe einer geraden Programmzähleradresse, holt sie



512 dyn. Abfrageverstärker für die Spaltenleitungen

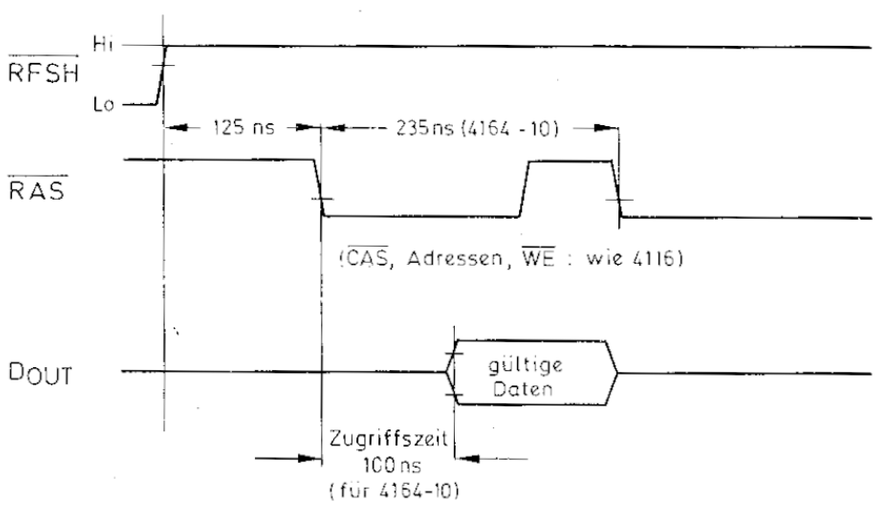


Bild 4.52

zwar alles, was der Bus bietet (D0 . . . D15), separiert aber intern auf das obere Byte D8 . . . D15. Der Programmzähler wird anschließend inkrementiert (um eine Stelle) und der nächste Holvorgang zielt bei der jetzt ungeraden Adresse auf das untere Byte D0 . . . D7 ab, gleichgültig, welchen Wert das obere Byte hat. Daraus folgt, daß bei einem Befehlsholzyklus eines Einwortbefehls insgesamt ein zweites Wort im Speicherwerk ungenutzt bleibt; man kann auch sagen, — verschenkt wird. In diesen Apfel muß man beißen, wenn ein Speicherwerk nach Bild 4.53 verwendet wird. Holt die CPU Daten an einer bliebigen Adresse (indirekt adressiert aus einem Allzweckregister), wird im Wortmodus nichts verschenkt. Es wird das geholte Wort CPU intern in der ganzen Breite D0 . . . D15 übernommen. Diesen Unterschied sollte man wissen, wenn man die besagte Schaltung anwendet.

Wer nicht bereit ist, auch nur ein Bit herzugeben, sollte sich im Prinzip nach der Schaltung in Bild 4.42 plus Bild 4.46 orientieren. Das bedeutet nichts anderes als daß man sein System im Bytekonzept streng nach den Regeln entwirft. Die

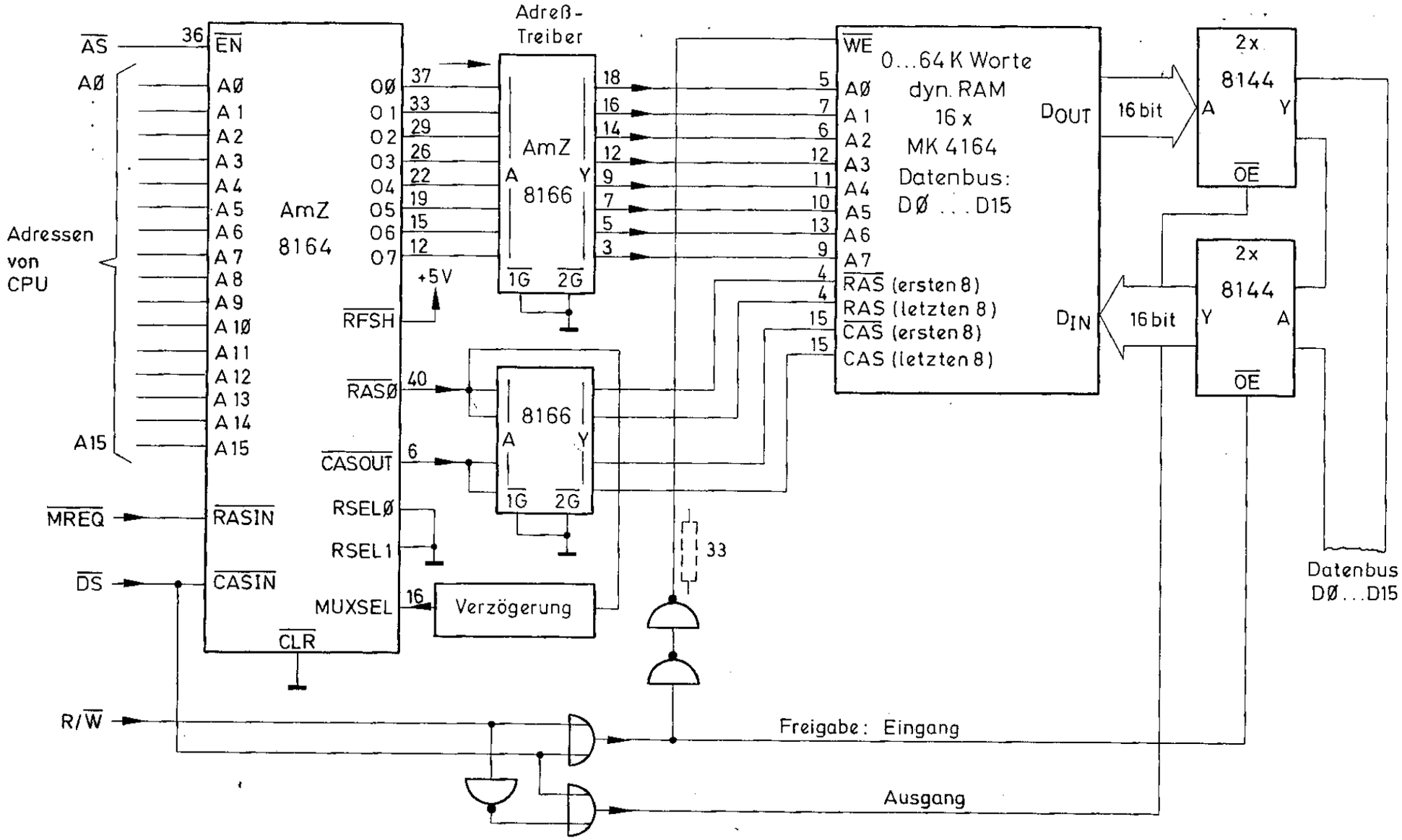

AS
AØ
Adressen von CPU
A15
MREQ
DS
R/W
36
EN
AØ
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
RASIN
CASIN
CLR
AmZ 8164
OØ
O1
O2
O3
O4
O5
O6
O7
37
33
29
26
22
19
15
12
+5V
RFSH
RASØ
40
CASOUT
6
RSELØ
RSEL1
MUXSEL
16
Verzögerung
Adreß-Treiber
AmZ 8166
A
Y
1G
2G
18
16
14
12
9
7
5
3
8166
33
WE
0...64 K Worte dyn. RAM 16 x MK 4164 Datenbus: DØ ... D15
5
7
6
12
11
10
13
9
4
4
15
15
RAS (ersten 8)
RAS (letzten 8)
CAS (ersten 8)
CAS (letzten 8)
DOUT
DIN
16 bit
16 bit
2x 8144
OE
Datenbus DØ...D15
Freigabe: Eingang
Ausgang

Bild 4.53

Schaltung in Bild 4.53 wird dann nicht mehr so einfach aussehen. Zwei Bytegruppen der Größe 64 KB, in unteres und oberes Byte getrennt, werden von zwei Zeilenfreigabeleitungen $\overline{RAS\ 0}$ und $\overline{RAS\ 1}$ angewählt. Eine UND-Verknüpfung beider RAS-Leitungen mit der nachfolgenden Verzögerung auf den MUXSEL-Eingang schaltet in bereits bekannter Art auf die Spaltenadresse um. Das Datenbus-Interface besteht aus vier AmZ 8144, von denen jeweils zwei getrennt als Ein- und Ausgangstreiber angesteuert werden.

Dem Leser, der den Speichertransfer verstanden und die letzten drei Seiten aufmerksam gelesen hat, muß eine Tatsache auffallen. 64 KB mit den 4164 RAMs bleiben 64 KB in acht Bausteinen, die sich nicht trennen lassen. Verwendet man 64K Worte (16 Bausteine), geht in jedem Fall Speicherplatz verloren, wenn im Bytemodus zugegriffen wird. Man kann 64 KB rechnerisch in gerade und ungerade Adressen teilen. Die Folge ist, daß das 64 KB-Speicherwerk mit 64K RAMssniemals im Wortmodus betrieben werden kann. Würde man ausschließlich Programme in dem 64 KB-Feld unterbringen, kommt man mit acht Bausteinen aus. Verlangt man aber nach einem kombinierten Programmcode- und Datensegment bei wechselndem Byte- und Wortzugriff, funktioniert das Speicherfeld mit 64 KB aus acht 64K-Typen nicht. Was bleibt nach diesen Überlegungen? Das Speicherwerk nach Bild 4.53 ist doch geeignet.

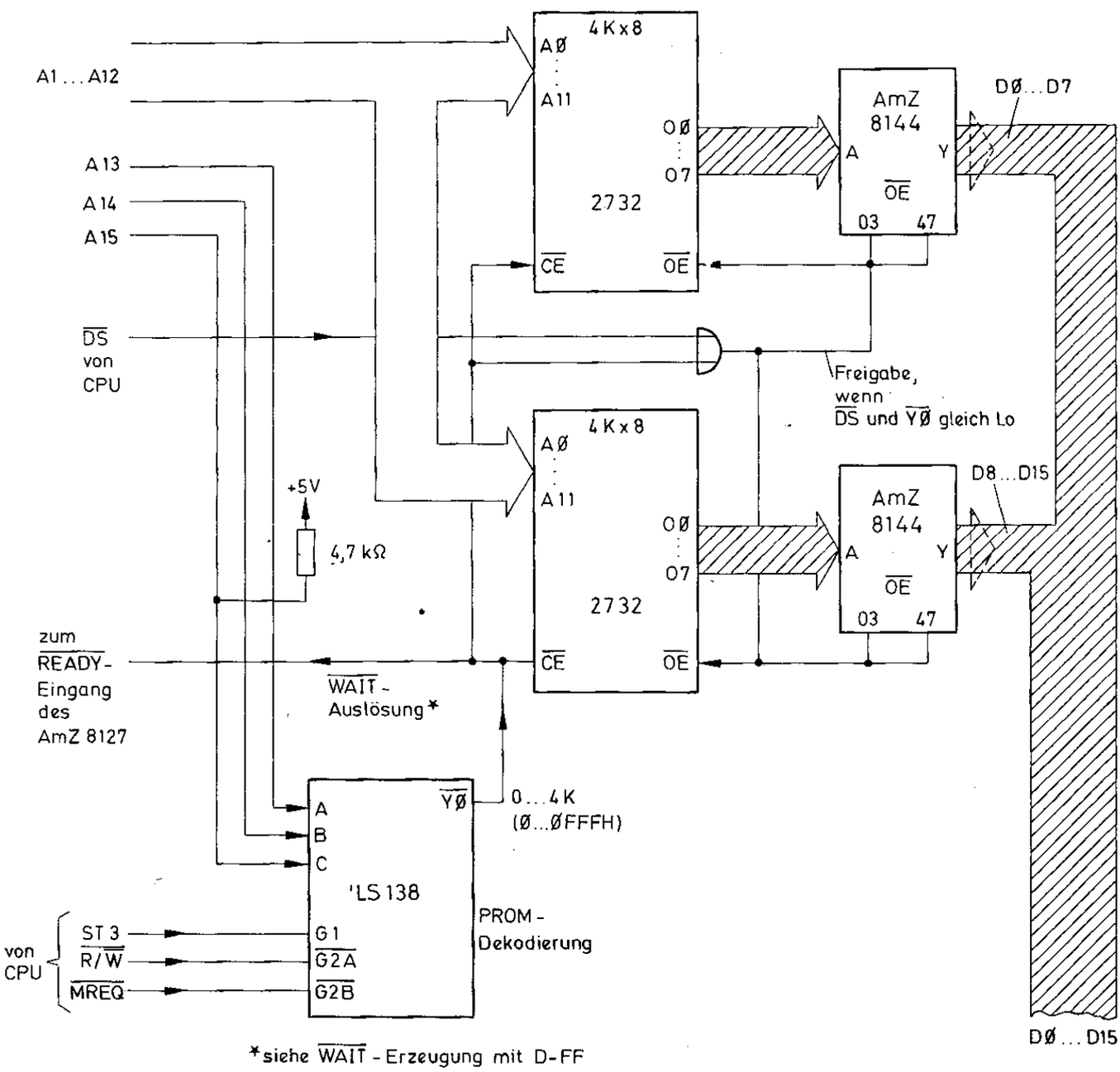
Die Schaltung des gezeigten Speicherwerks benötigt sicher keine weitere Erklärung, weil es sehr durchsichtig ist. R/$\overline{W}$ bzw. $\overline{WE}$ zu den Speicherbausteinen sollte nach den Gattern mit einem 33 Ω-Widerstand versehen werden oder man benützt noch die vier verbleibenden Leitungstreiber eines AmZ 8166.

Abschließend sei zum Abschnitt 4.4 bemerkt, daß dynamische Speicher generell kritischer in der Anwendung gegenüber statischen Typen sind. Beachtet man jedoch eine sehr kurze Leiterbahnführung der Adreß- und Steuerleitungen, geradlinige breite Stromversorgungsleitungen und ganz besonders das kapazitive Entkoppeln der Stromzuführungen (DC-Leitungen) jedes Bausteins, ist der Betrieb erfolgreich und statischen Speicherwerken überlegen. Bei kritischen Fällen müssen die Bausteine auch fest auf die Platine gelötet werden.

Der Z 8000 ist für den Einsatz dynamischer Speicher prädestiniert. Man sollte sie als Standardspeicher fest in das System eingliedern.

4.5 ANSCHLUSS-SCHEMATIK FÜR FESTWERTSPEICHER

Die Interfacelogik zum Betrieb von Festwertspeichern in einem Z 8000-System ist in jeder Hinsicht einfacher als die für RAMs. Generell ist der Betrieb von Festwertspeichern nur mit einem getrennten Adreß-/Datenbus möglich. **Bild 4.54**



*siehe WAIT - Erzeugung mit D-FF

Bild 4.54

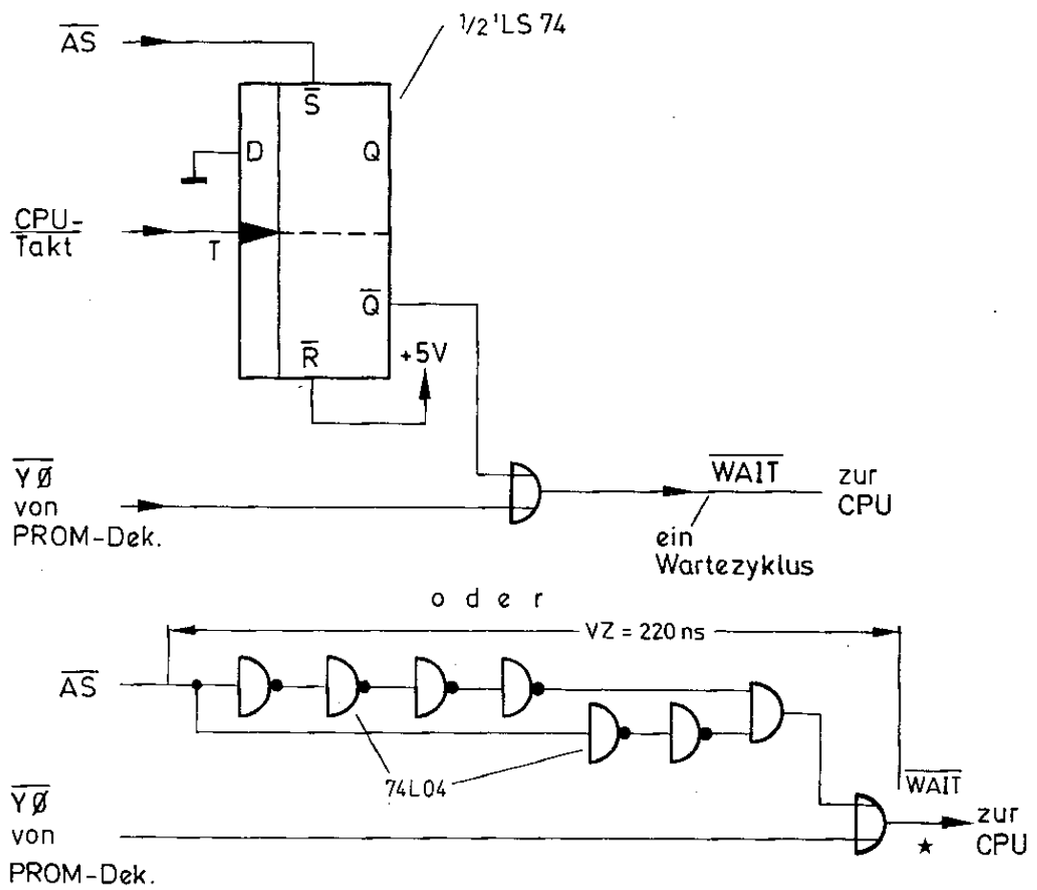
zeigt das komplette Interface mit Dekodierung und Datentreibergatter. Die in diesem Bild bezeichneten EPROMs des Typs 2732 mit 4 KB Kapazität enthalten ein Monitorprogramm für ein Z 8000-System, in dem elementare Routinen enthalten sind. Daten einlesen über die Tastatur; Ausgabe der Registerinhalte, Statusfunktionsanzeige und einiges mehr gehören zu solchen elementaren Routinen. Wie aus der Dekodierung zu ersehen ist, wird von der absoluten Nulladresse aufwärts der Speicher auslesbar. Das ist als Bedingung zu beachten. Erinnert man sich in Kapitel 2 (Bild 2.41) an die Ursache, bekommt man die Antwort darauf. Wird ein CPU-Reset durchgeführt, erscheint die Adresse 0002H am Bus. Die CPU wünscht den FCW-Wert einzulesen. Die Folgeadressen sind 0004H und 0006H (nur Z 8001). Findet die CPU nicht an dieser Stelle, was sie sucht, gibts Ärger. Man muß sicherstellen, daß die CPU an diesen Speicherstellen einen definierten Wert zu ihrer Initialisierung findet, damit sie in einer vorhersehbaren Form in Aktion tritt.

Betrachtet man das Datenbus-Interface, fällt auf, daß bei jedem Lesevorgang oberes und unteres Byte gleichzeitig ausgelesen wird. Man kann das mit ruhigem Gewissen so lassen, weil die CPU beim Lesen einer geraden bzw. ungeraden Adresse die Separierung intern nach oberem bzw. unterem Byte selbst vornimmt. Wer es nicht glaubt, kann mit der Adreßleitung A0 eine externe Separierung vornehmen. Es wird nichts bringen. die Mühe wird umsonst sein.

Überwiegend haben auch Festwertspeicher eine passive Betriebsart, d.h. wird der $\overline{CE}$-Eingang des Bausteins nicht aktiviert, benötigt er meist nur ein Viertel seiner üblichen Leistung im aktiven Zustand. Deshalb sollte die Chipfreigabe aus der Dekodierleitung Y0 angesteuert und nicht statisch auf Lo gelegt werden. Das spart Strom. Bei dieser Maßnahme ist jedoch für EPROMs ein Wartezyklus in jedem Fall notwendig, wenn die CPU mit einem 4 MHz-Takt arbeitet. 290 ns Zugriffszeit ist gerade die Grenze, die einen Transfer ohne Wartezyklus noch zuläßt. 300 ns EPROMs und die genannte Betriebsart lassen das nicht zu. Wie in Bild 4.54 dargestellt ist, wird eine Wartezyklus-Anforderung dem CPU-Taktgenerator-Baustein AmZ 8127 am $\overline{READY}$-Eingang übermittelt, der von sich aus den $\overline{WAIT}$-Eingang der CPU ansteuert (siehe Bild 4.3). Das ist die einfachste Erzeugung eines Wartezyklus. Hat man diesen Baustein nicht zur Verfügung, muß ein Wartezyklus diskret erzeugt werden. In **Bild 4.55** wird dargestellt, welche Möglichkeiten man dazu hat (zwei von vielen).

Die Erzeugung durch ein Flipflop LS 74 und eine angehängte ODER-Verknüpfung ist die gebräuchlichste Methode. Während der Adreßausgabe wird durch $\overline{AS}$ gleich Lo an $\overline{S}$ der $\overline{Q}$-Ausgang des Flipflops Lo gesetzt. Der am T-Eingang angelegte invertierte CPU-Takt bewirkt an der Lo ↑ Hi-Flanke im ersten Takt keine Änderung des Zustandes (weil $\overline{AS}$ noch Lo ist), d.h. $\overline{Q}$ bleibt Lo. Erst an der Lo ↑ Hi-Flanke des zweiten Takts wird $\overline{Q}$ gleich Hi. Zu diesem Zeitpunkt wurde bereits der $\overline{WAIT}$-Eingang abgefragt, der über das ODER durch $\overline{Y0}$ gleich Lo auch an $\overline{WAIT}$ Lo war. Damit wird ein Wartezyklus eingefügt. Die zweite Methode mit der Verzögerung funktioniert genauso sicher, jedoch mit der Bedingung, daß der CPU-Takt nicht unter 250 ns Zykluszeit liegt.

Die Aktivierung des Chipselekt-Signals am Y0-Ausgang des Adreßdekoders LS 138 ist von dem Wert der drei Adreßeingänge A . . . C und den drei Bedingungseingängen abhängig. Die Invertierung des Lesesignals $R/\overline{W}$ ergibt Lo beim Lesen. $\overline{MREQ}$ ergibt Lo bei einem Speicherzyklus. Wird die Statusleitung ST3 Hi, zeigt das generelle Speicheroperationen an. Man kann z.B. auch eine Verknüpfung der Statusleitungen ST2 und ST3 bilden, woraus sich die Anwahl eines reinen Programmcodespeichers ergibt (als Ersatz für $\overline{MREQ}$). Das liegt im Ermessen des Hardwaredesigners.


AS
1/2 'LS 74
S
D
Q
CPU-
Takt
T
Q
R
+5V
YØ
von
PROM-Dek.
WAIT
zur
CPU
ein
Wartezyklus
o d e r
VZ = 220 ns
AS
74L04
WAIT
YØ
von
PROM-Dek.
zur
CPU
★

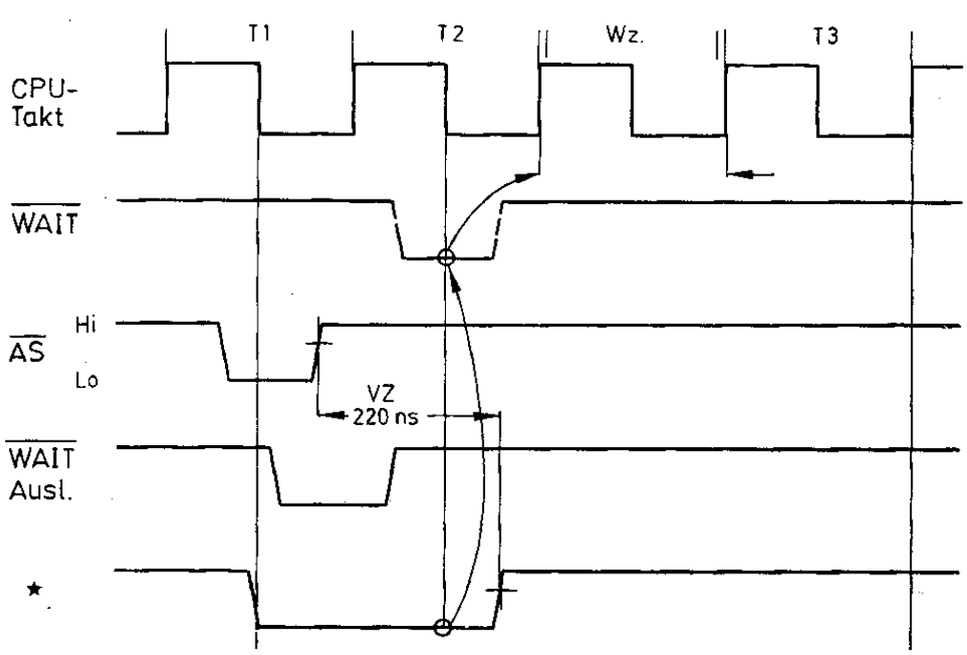

T1
T2
Wz.
T3
CPU-
Takt
WAIT
AS
Hi
Lo
VZ
220 ns
WAIT
Ausl.
★

Bild 4.55

4.5.1 Unterschiede bei der Anwendung von EPROMs und bipolaren PROMs

Im Prinzip kann als Festwertspeicher für den Z 8000 alles verwendet werden, was auf eine Adressierung einen definierten digitalen Pegel lierfert. Mit UV-Licht löschbare Festwertspeicher werden sicher in überwiegender Zahl einen Einsatz verzeichnen. Sie erfordern das Einschieben eines Wartezyklus bei einer Leseoperation des Z 8000. Bei den durchweg extrem schnellen bipolaren PROMs ist dies nicht erforderlich. Dieser Speichertyp hat im Vergleich zu EPROMs noch nicht die Dichte erreicht, so daß mehr Bausteine für ein Festwertspeicherfeld erforderlich sind. Die Zuverlässigkeit bipolarer PROMs ist jedoch anerkannt außergewöhnlich hoch. Ihr Preis allerdings auch.

Die folgenden **Bilder 4.56, 4.57** und **4.58** zeigen eine Auswahl moderner Hochleistungs-Festwertspeicher von 2 KB . . . 8 KB EPROMs und 1 KB, 2 KB und 2K x 4 bit bipolaren PROMs.

UV löschbare EPROMs (Festwertspeicher)
Zugriff 300. . .450 ns ; bedingt Wartezyklus der CPU

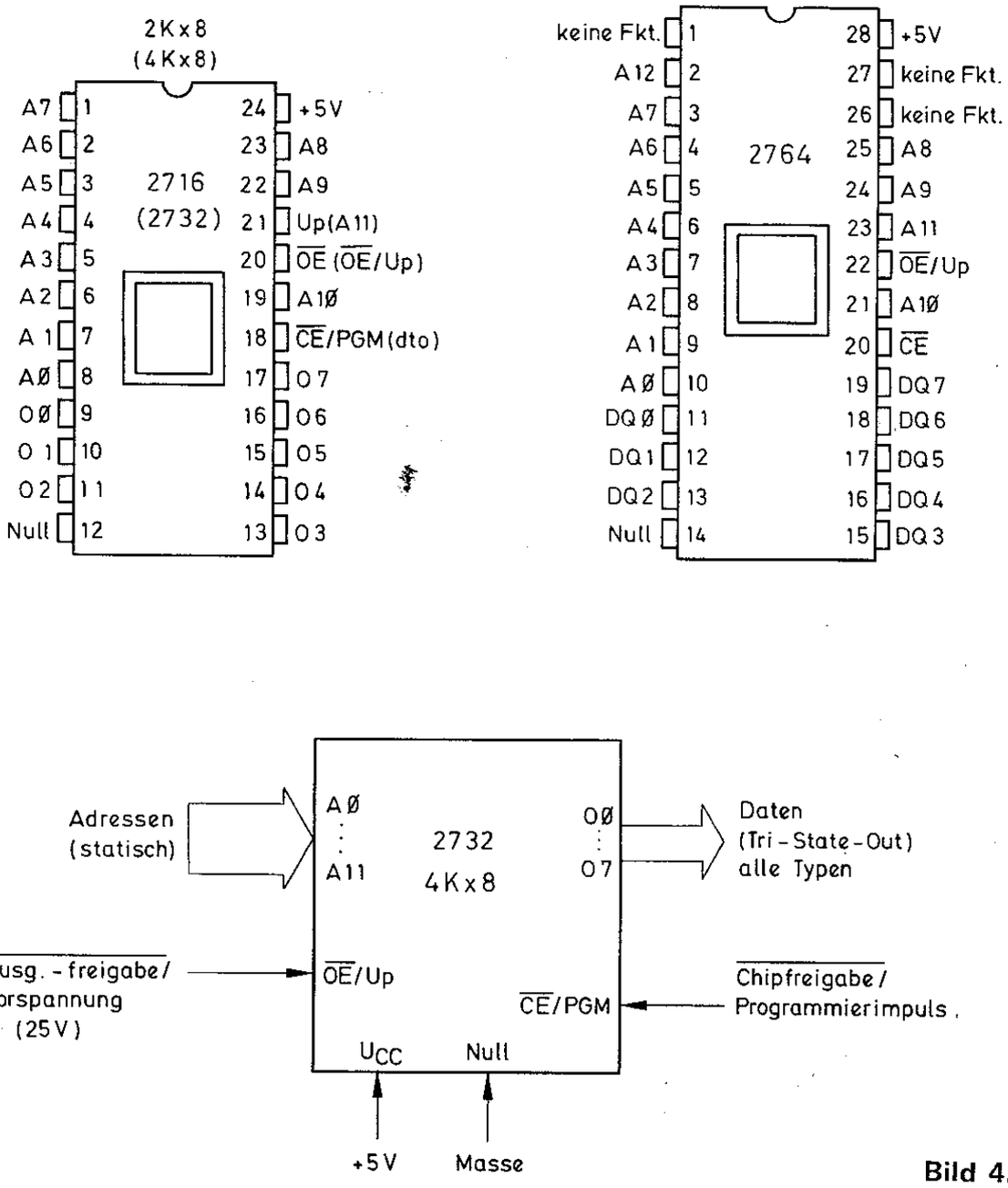
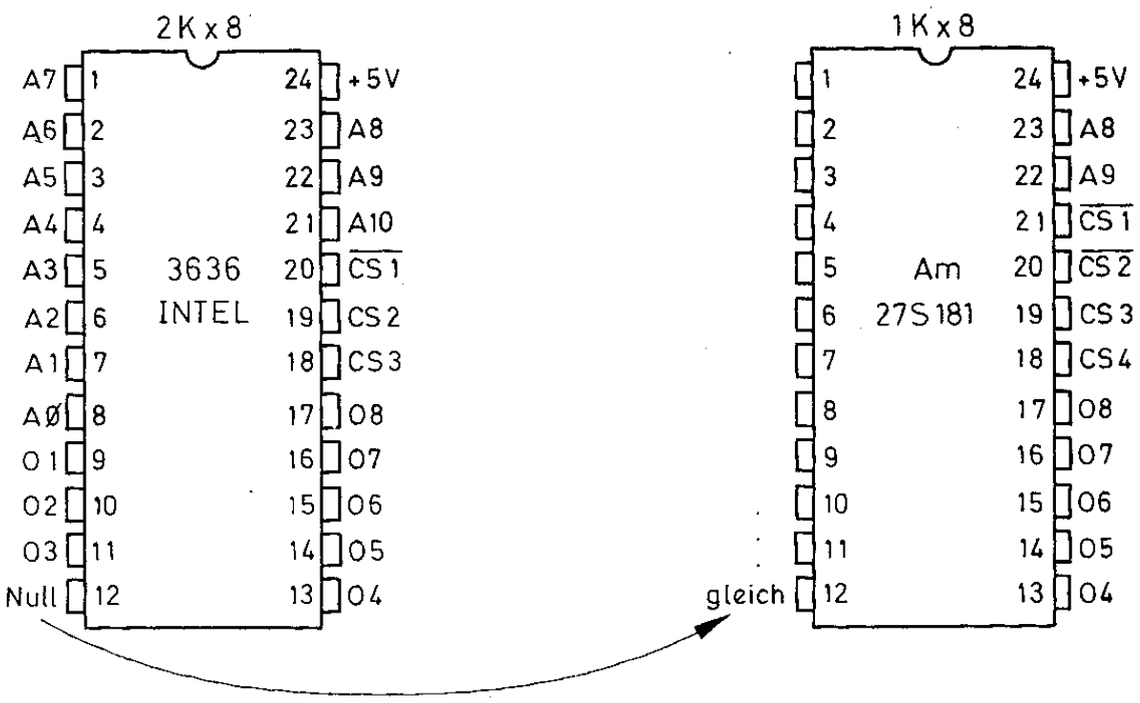


Bild 4.56

Bipolare PROMs (Festwertspeicher)
Zugriff 60...80 ns; Vorteil! benötigen keinen
Wartezyklus von der CPU



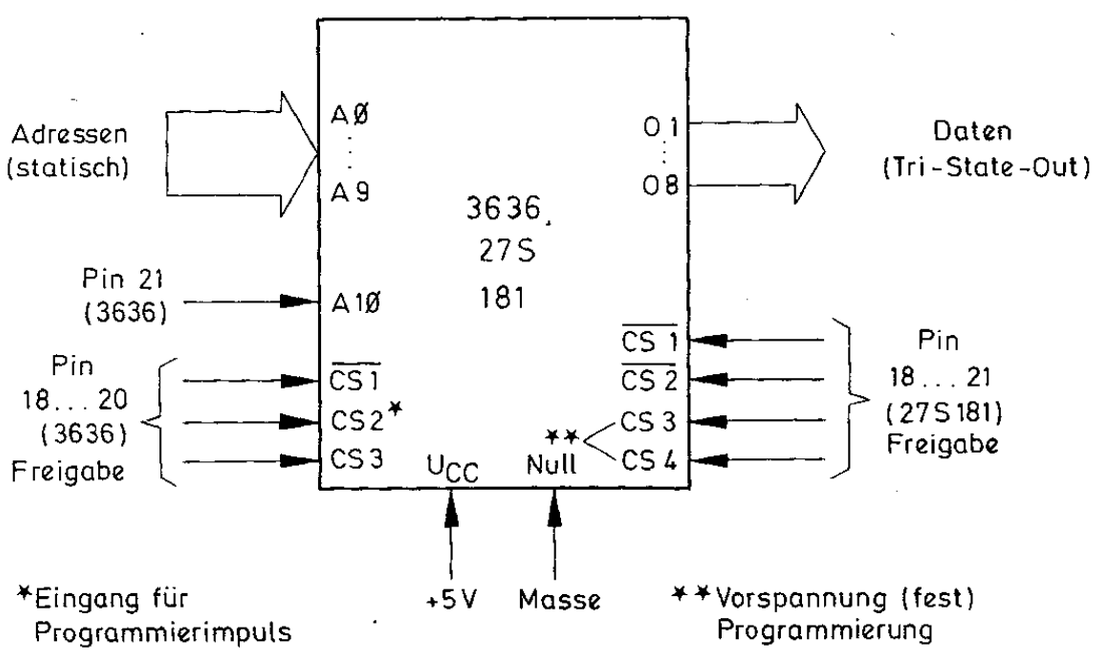


Bild 4.57

Bipolares PROM
Zugriff 40 ns ; kein Wartezyklus nötig

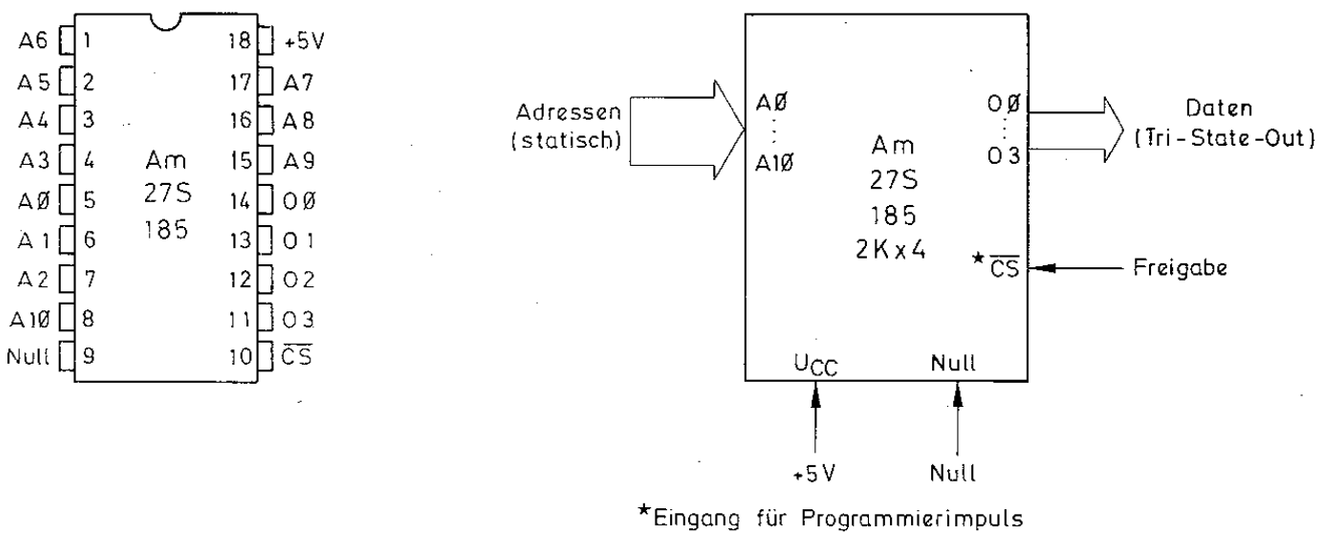


Bild 4.58

4.6 ANSCHLUSS-SCHEMATIK FÜR GÄNGIGE E/A-BAUSTEINE MIT 8-BIT-BUS

Die Bestrebungen von Zilog und AMD in gleicher Weise richten sich auf ein Bausteinsystem parallel zur CPU, das weitgehende Anpassung an das Signalverhalten des Prozessors zeigt. Einige Beispiele wurden bereits gegeben; weitere folgen in Kapitel 10. Andererseits sind die Steuersignale des Prozessors so transparent und logisch, daß der Aufbau von E/A-Schnittstellen für einen Hardwaretechniker in absolut angemessenem Rahmen zu erstellen ist. Die Bedeutung von Schnittstellen hat oberste Priorität. Über einen E/A-Port bekommt die CPU ihre Aufgaben gestellt und wird ebenso über eine Schnittstelle bezüglich ihrer Funktion überprüft.
In diesem Abschnitt wird ein Interface mit den Bausteintypen Am 9555A (bzw. Am 8255A) und der Z 80A PIO

besprochen. Bereits zu Anfang dieses Kapitels wurde in Bild 4.5 ein Interface zu dem Parallel-Port Am 2950 vorgestellt. Dieser bidirektionale 8-bit-E/A-Baustein ist eine vollwertige Schnittstelle, die nur ein paar Gatter zur Steuerung vom Z 8000 aus benötigt. Bei diesem Interface war von einem Einadreß-Port die Rede, weil eine einzige Adresse aus dem 64K-Feld separiert wurde, um die Freigabe für einen E/A-Transfer zu erzielen. Wollte man den Baustein Am

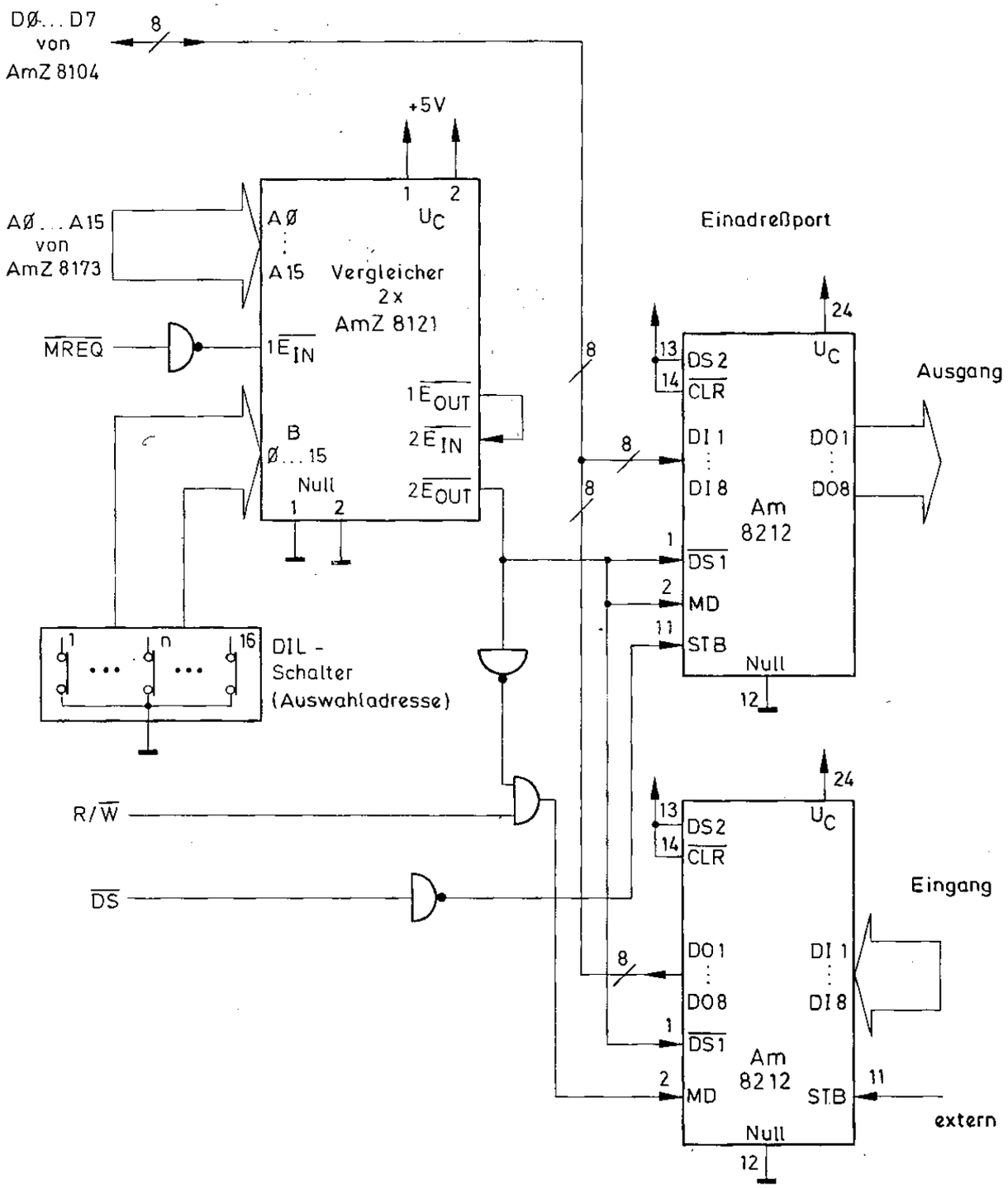


Bild 4.59

2950 nochmal reduzieren, ist damit die Urform eines Einadreß-Ports angesprochen. Der angesprochene Baustein des Typs 8212 ist entweder ein Ein- oder Ausgangs-Port, je nachdem, wie er verdrahtet wird. Er enthält 8 D-Flipflops, an dessen Q-Ausgängen Tri-State-Treiber angeschaltet sind. **Bild 4.59** zeigt diese Urform eines bidirektionalen Einadreß-Ports mit zwei Am 8212, die auf sehr einfache Art an die Basis-Konfiguration eines Z 8000-Systems angeschlossen werden kann. An den 16-bit-DIL-Schalter wird die Adresse für die Freigabeerteilung eingestellt. Es erübrigt sich, diese Schaltung näher zu erklären, weil sie mehr exemplarischen Wert besitzt. Einen Port auf diese Weise aufzubauen ist zwar möglich, aber technisch überholt.

Ein weitaus leistungsfähigerer Schnittstellen-Baustein ist der Am 9555A (8255A). **Bild 4.60** zeigt das komplette Interface zum Z 8000. Eine sehr geringe Zahl zusätzlicher Gatter sorgt für den Anschluß von der CPU her. Die Bezeichnung Einadreß-Port ist hier nicht mehr ganz zutreffend, weil die drei unabhängigen 8-bit-Ports vier Schnittstellenadressen der CPU erfordern. Über das Schreib-/Lesesignal läßt sich die Richtung des Transfers festlegen. Die E/A-Adressen liegen alle hintereinander. An den Vergleicherbausteinen AmZ 8121 werden von der 16-bit-Adresse der CPU an die A-Eingänge die beiden untersten Adreßleitungen A0 und A1 zugleich an die B-Eingänge angelegt. Die Auswahladresse ist deshalb nur noch 14 bit breit und läßt sich in bekannter Weise an den DIL-Schaltern einstellen. Die Adreßleitungen A0 und A1 werden zugleich an die gleichnamigen Anschlüsse des Schnittstellen-Bausteins gelegt. Bei der Adressierung der einzelnen Portgruppen sind über das Interface folgende Permutationen von A0, A1, $\overline{WR}$ und $\overline{RD}$ möglich.

A0	A1	$\overline{RD}$	$\overline{WR}$	Operationen über D0 . . . D7	
0	0	Lo	Hi	Port A → Datenbus	auslesen des jeweiligen Port-Registers
1	0	Lo	Hi	Port B → Datenbus	
0	1	Lo	Hi	Port C → Datenbus	
0	0	Hi	Lo	Port A	Byte nach außen ausgeben
1	0	Hi	Lo	Datenbus → Port B	
0	1	Hi	Lo	Port C	
1	1	Hi	Lo	Steuerwortregister (8 bit) laden	≪ lesen nicht möglich ≫

Die obige Konfiguration setzt voraus, daß der Chipselekt-Eingang $\overline{CS}$ am Port-Baustein gleich Lo ist. Wird er Hi, ist der Datenbusanschluß D0 . . . D7 im hochohmigen Zustand. Ferner ist die Permutation A0 = A1 = Hi, $\overline{RD}$ = Lo, $\overline{WR}$ = Hi und $\overline{CS}$ = Lo nicht erlaubt, weil das Steuerregister nicht gelesen werden kann. Mit folgendem Befehl des Z 8000 ist ein Transfer möglich:

INB RL0,@R 10

übernimmt ein Byte in das Byteregister RL0 von einem adressierten Port. Bei diesem Befehl in indirekter Adressierung muß die Portadresse **unbedingt ungerade** sein, weil es ein Bytebefehl ist und das untere Byte D0 . . . D7 gelesen wird. Wendet man hingegen den "Input"-Befehl folgendermaßen an:

IN R 0,@R 10

ist es unerheblich, ob die Adresse gerade oder ungerade ist, weil D0 . . . D15 gelesen wird. Das Zielregister R0 ist auch ein Wort-Register. Man kann auch direkt

adressieren (Portadresse als 16-bit-WErt im Befehl inbegriffen):

```
  IN   R 0,%004E
  INB  RL0,%0049  ≪ungerade≫
 OUT  %004F,R1
OUTB  %004F,RL1
```

Die "Output"-Befehle können ebenso indirekt geschrieben werden:

```
  OUT  @R 10,R1
 OUTB  @R 10,RL1
```

Die Adresse in R 10 für die Byteausgabe muß nicht unbedingt ungerade sein, weil die CPU das Byte auf beide Bushälften dupliziert. Die schaltungstechnische Maßnahme, daß $\overline{WR}$ oder $\overline{RD}$ am 9555 erst bei $\overline{DS}$ gleich Lo erzeugt wird, ist hinreichend für den Datentransfer und verursacht keine zeitlichen Schwierigkeiten, weil die CPU bei einem E/A-Transfer grundsätzlich einen Wartezyklus einfügt. Ein E/A-Transfer mit dem Portbaustein 9555A ist eine absolut sichere und sehr einfach zu behandelnde Angelegenheit.

Will man ernsthaft 16-bit-Daten transferieren, bleibt in der Tat keine andere hardware-technische Möglichkeit, als mit zwei Bausteinen z.B. des Typs Am 2950 zu arbeiten. Jeweils ein Baustein wird mit dem Datenbus an den A-Anschlüssen auf das untere Byte und auf das obere Byte verbunden. Soll im Wortformat nur einseitig übertragen werden, kann auch das 8-bit-D-Flipflop AmZ 8120 verwendet werden. Eine 16-bit-Datenschnittstelle dieser Art wäre wieder als Einadreß-Port zu bezeichnen.
In technischer Hinsicht bietet die zweifach bidirektionale Parallelschnittstelle Z 80A PIO mehr als die bisher besprochenen. Die Interruptfreigabeleitungen IEI - IEO (Erklärung siehe Abschnitt 2.9) erlauben einen verketteten Interrupt-Anforderungsmechanismus auf einer Ebene mit stetig fallender Priorität. Die extern ansteuerbaren Strobe-Eingänge und die dazugehörigen Quittungssignal-Ausgänge (Ready) für jeden der beiden Ports ermöglicht eine gezielte Interruptbehandlung. Das 8-bit-Maskierungsregister erlaubt die Festlegung eines Ein- oder Ausgangs eines Bits der jeweiligen Portanschlüsse.

Jedes Ding hat normalerweise und bekanntlich zwei Seiten, – so auch hier. Die Z 80A PIO wurde zu einem Zeitpunkt entworfen, als von einem Z 8000 noch nicht die Rede war. Die PIO wurde logischerweise auf die Z 80-CPU zugeschnitten, was sicherlich zu verstehen ist. Die Z 80-CPU hat zwei Systemsteuerausgänge, die beim Z 8000 prinzipiell nicht zu finden sind. Es handelt sich hier um die Signale $\overline{IORQ}$ (Anzeige einer Schnittstellenadresse am unteren Byte A0 . . . A7 des Adreßbus; Anzeige eines E/A-Vektors am Datenbus, ausgegeben von einem Interrupt auslösenden Baustein und $\overline{M1}$ Maschinenzyklusanzeige; während des ersten Befehlholzyklus zwei Takte < ohne Wartezyklus > aktiv) $\overline{M1}$ ist auch aktiv, wenn die CPU einen Interrupt-Anerkennungszyklus durchläuft. Und genau dieses Signal verlangt die Z 80A PIO von der Z 80-CPU, was sie auch von der CPU direkt bekommt. Was bleibt hier übrig, als diese Signale für einen Anschluß des Z 8000 an die PIO durch externe Logik zu simulieren. Einen Z 8000-Maschinenzyklus kann man relativ leicht erzeugen. **Bild 4.61** zeigt das Signaldiagramm eines Maschinenzyklus der CPU, von welchen Flanken er zu definieren ist. Der Z 8000 sendet mit der Hi ↓ Lo-Flanke von $\overline{AS}$ die Information aus, daß sich in Kürze (etwa 20 . . . 30 ns später) eine Adresse am AD-Bus etabliert hat. Die andere wichtige Flanke ist die Lo ↑ Hi-Flanke von $\overline{DS}$. Nach dieser Flanke am CPU-Ausgang müssen die Daten bereits übernommen sein. Die Betonung liegt auf Holzyklus. $\overline{M1}$ bezieht sich also generell auf einen Lesevorgang. Ist $\overline{DS}$ gleich Hi geworden, nimmt die CPU keine Daten mehr an. Somit sind die Grenzen abgesteckt: Hi ↓ Lo von $\overline{AS}$ und Lo ↑ Hi von $\overline{DS}$. Ein konstantes Lo-Signal während dieser Zeit entspricht diesem $\overline{M1}$-Maschinenzyklus, der die erste Phase eines Befehlszyklus darstellt. An den Statusausgängen muß für den $\overline{M1}$ der Status Nr. 13 (IF 1) erscheinen. Mit diesem Status würde der

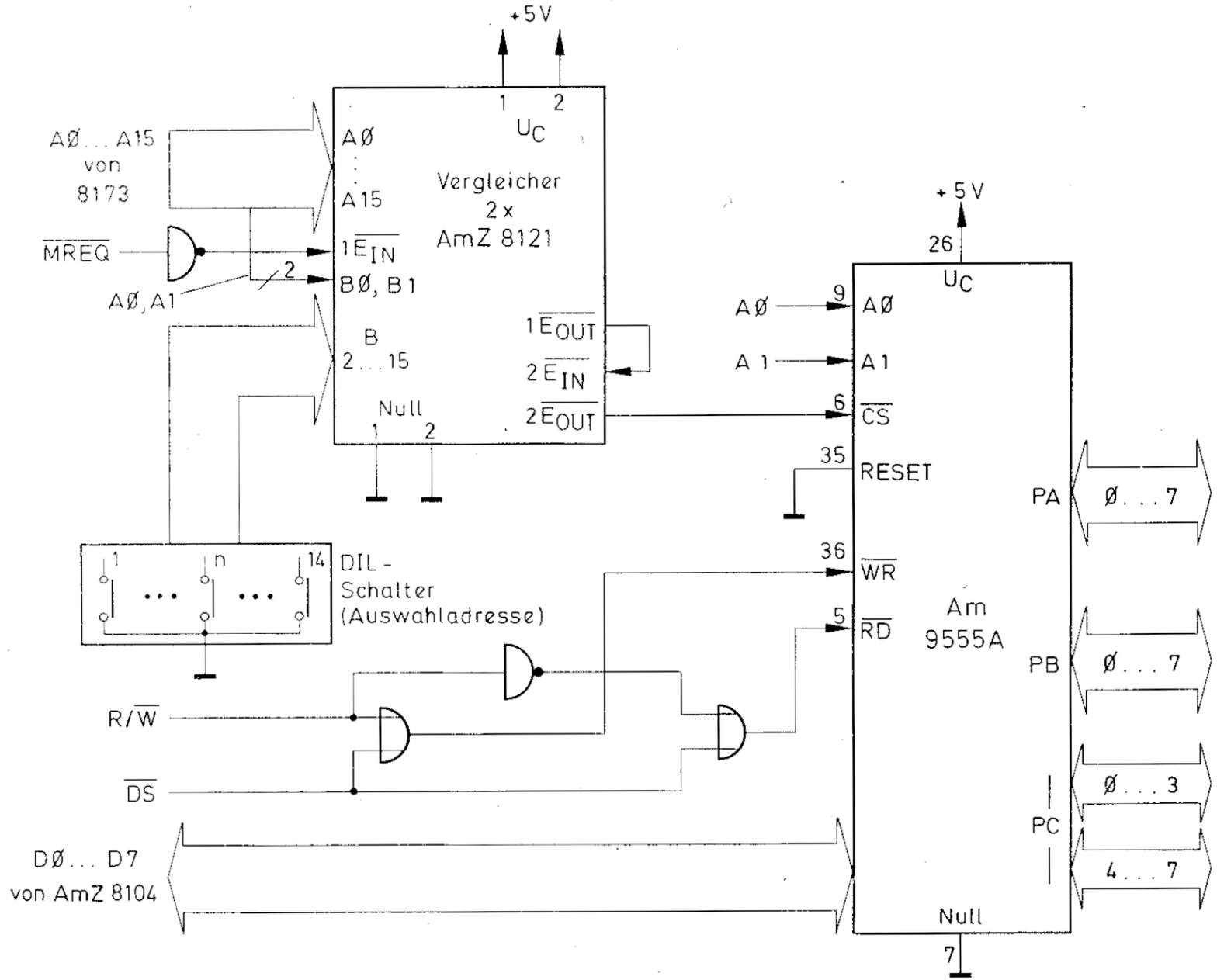

+5V
1
2
UC
AØ...A15
von
8173
AØ
A15
Vergleicher
2x
AmZ 8121
MREQ
1EIN
2
BØ, B1
AØ, A1
B
2...15
1EOUT
2EIN
2EOUT
Null
1
2
+5V
26
UC
AØ
9
AØ
A1
A1
6
CS
35
RESET
PA
Ø...7
36
WR
5
RD
Am
9555A
PB
Ø...7
PC
Ø...3
4...7
Null
7
1
n
14
DIL-
Schalter
(Auswahladresse)
R/W
DS
DØ...D7
von AmZ 8104

Bild 4.60

Erzeugung eines Z80 kompatiblen M1
aus den Z8000-Signalen

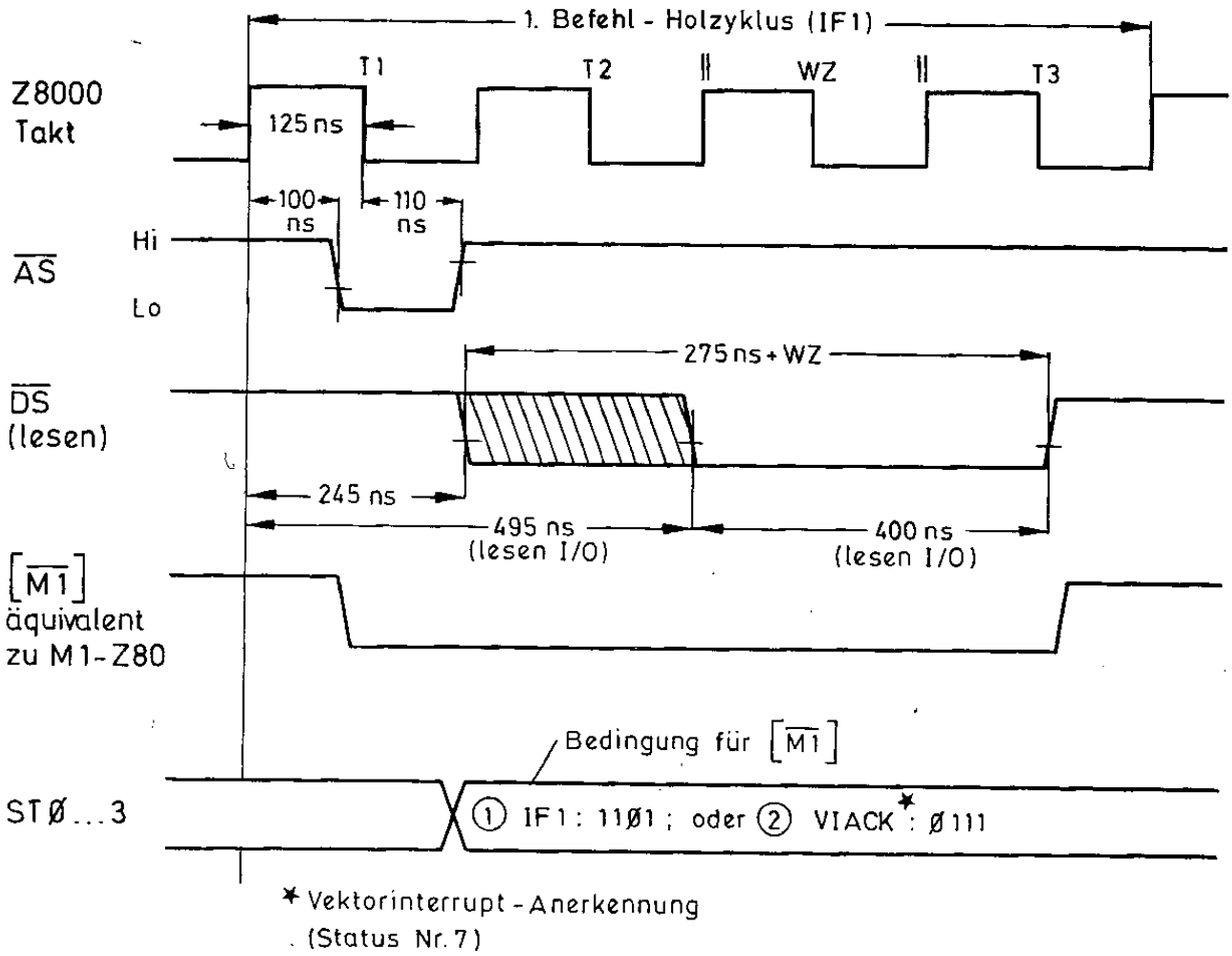
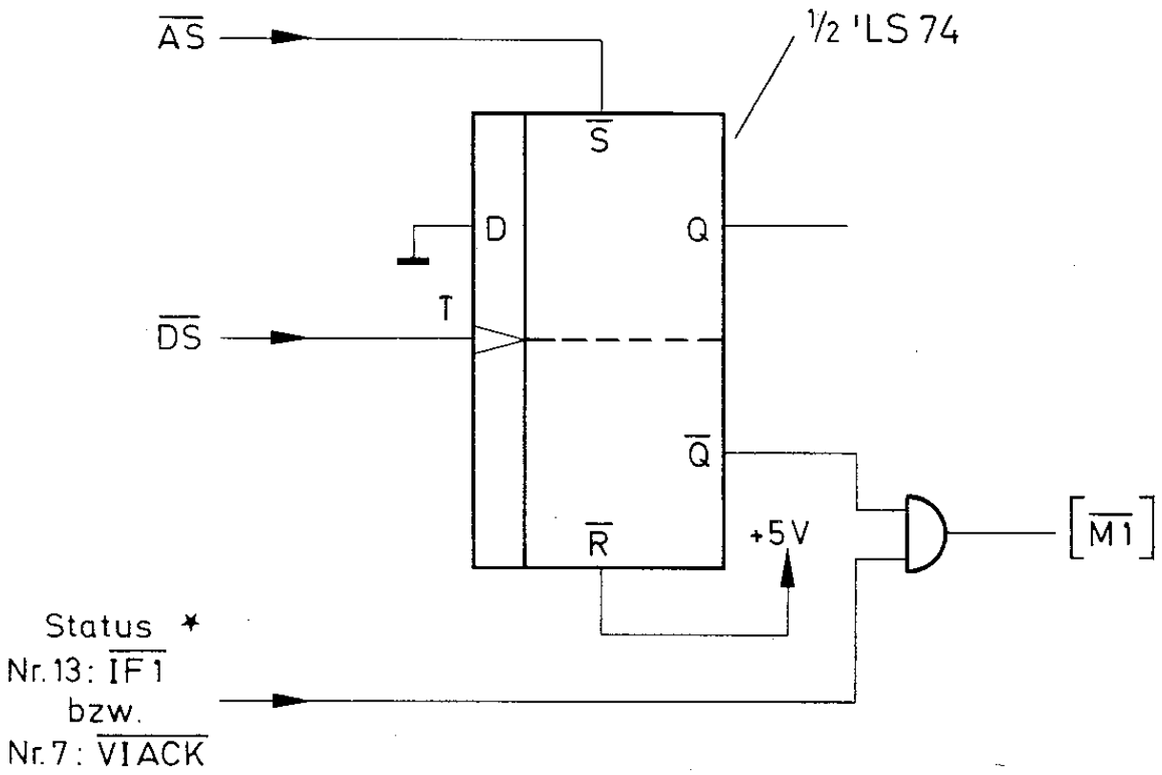


Bild 4.61

erste aktive Zustand simuliert, den der $\overline{M1}$-Zyklus der Z 80-CPU anzeigt. Der eigentlich benötigte Fall für die PIO ist der Empfang der Kombination von $\overline{M1}$ und $\overline{IORQ}$ im aktiven Zustand. Von der PIO wurde ein Interrupt angefordert, der von der CPU einer Anerkennung bedarf. Der Z 80-Anerkennungszyklus zeigt die Aktivität von $\overline{M1}$ und $\overline{IORQ}$, wobei, während $\overline{IORQ}$ gleich Lo ist, ein besagter Interruptvektor am Datenbus von der CPU gelesen wird. Genau diesen Fall muß die externe Logik für ein Port-Interface zwischen der Z 8000-CPU und der Z 80A PIO nachbilden. In **Bild 4.62** ist die einfache Erzeugung eines $\overline{M1}$ äquivalenten Zyklus dargestellt, der mit den Z 8000-Signalen $\overline{AS}$ und $\overline{DS}$ und einem LS74 erzeugt wird. Diese D-Flipflop-Schaltung ist funktionell völlig identisch



*für Anschluß des Z80A P10
wird Status Nr. 7 verwendet

Bild 4.62

mit einer Wartezykluserzeugung nach Bild 4.55. Die Hi↓Lo-Flanke von $\overline{AS}$ setzt Q gleich Hi und $\overline{Q}$ entsprechend Lo. Bei der nächsten Lo↑Hi-Taktflanke (in diesem Fall ausgelöst von $\overline{DS}$) wird $\overline{Q}$ wieder Hi. Der Verlauf an $\overline{Q}$ des LS74 ist mit $\overline{M1}$ identisch. Verknüpft man den Status Nr. 7 mit dem $\overline{Q}$-Ausgang wie gezeigt, hat man das gesuchte Z 80 äquivalente $\overline{M1}$-Signal.

Eine weitere Bedingung für die PIO muß gebildet werden. Die Sperrleitung IEO für eine Interruptfreigabe niedriger priorisierter PIOs (oder anderer Bausteine) bleibt so lange auf Lo (sperren), bis von der Z 80-CPU der Befehl RETI (RETurn from Interrupt → Rücksprung aus Interruptserviceroutine) bearbeitet wurde. Beide Bytes des Befehls werden am Datenbus von der PIO aufgenommen und intern dekodiert. Sie bewirken das Rücksetzen des internen IUS-Flags und das Deaktivieren von IEO (gleich Hi). Diese Funktion hat die Z 8000-VPU ebenfalls nicht in ihrem Repertoire. Folglich muß auch diese Funktion des Z 80 simuliert werden. Der sedezimale Opcode ED4DH für den RETI-Befehl bewirkt mit EDH das Deaktivieren von IEO und 4DH den Reset des IUS-Flags. Die hardwaremäßige Rückwirkung auf diese nicht vorhandene Softwarefunktion des Z 8000 zeigt sich in **Bild 4.63.** Am Schluß jeder Interruptserviceroutine muß der Z 8000 einen Scheinport ansprechen und auf den Datenbus die beiden Bitkombinationen EDH und 4DH hintereinander anlegen, damit die PIO durchführen kann, was sie durchführen muß. Das kleine Programm als Ersatz für die RETI-Operation des Z 80 kann folgendermaßen aussehen:

```
    LD  R0, %ED4D
OUTB  "RETI-Port", RH0
OUTB  "RETI-Port", RL0
```

Die Verwendung der direkten Adressierung ist von Vorteil. "RETI-Port" ist eine E/A-Adresse, die eine Aktivierung von Y6 am AmZ 8136 auslöst. "RETI-Port" muß folglich die Adresse 009CH ... 009FH haben, damit Y6 aktiv wird. Da der Baustein AmZ 8136 ein simulierter Port ist, muß $\overline{MREQ}$ gleich Hi sein. Die gesamte Schaltung eines PIO-Anschlusses an den Z 8000 zeigt **Bild 4.64.** Hier werden die bereits besprochenen Signale $\overline{M1}$ und Y6 mit einbezogen. Die weitere Logik erzeugt $\overline{IORQ}$ und $\overline{RD}$. Mit diesem Interface kann die Z 8000-CPU mit ihren E/A-Befehlen Bytes von außen übernehmen oder nach außen senden (jeweils Port A oder Port B), die PIO initialisieren, eine Interrupt-Anforderung behandeln und die PIO mit "RETI" deaktivieren. Somit gilt in dieser Konfiguration die Z 80A PIO als sehr brauchbare Parallelschnittstelle in einem Z 8000-System.

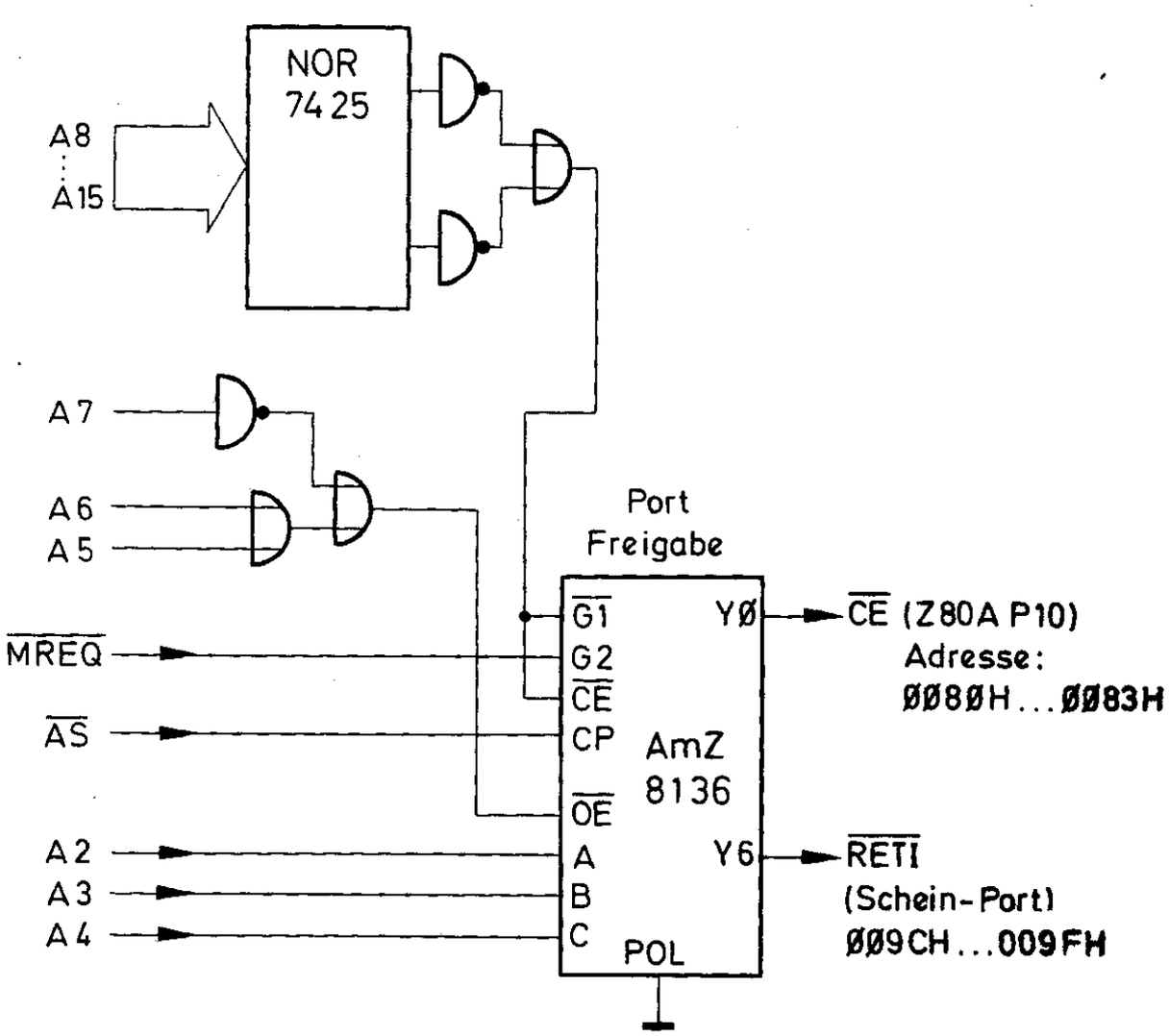


Bild 4.63

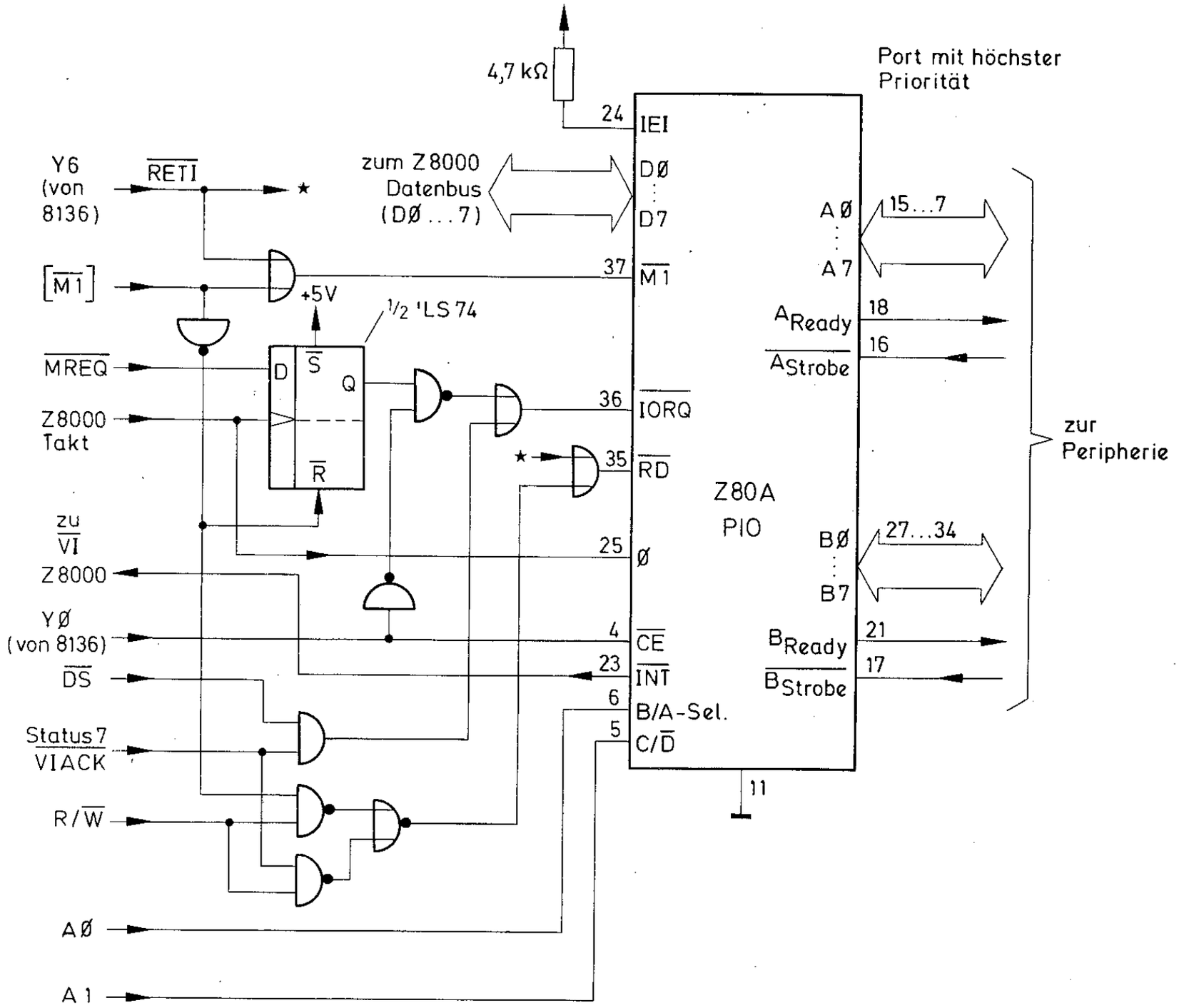

4,7 kΩ
Port mit höchster Priorität
24
IEI
Y6 (von 8136)
RETI
zum Z8000 Datenbus (DØ...7)
DØ
D7
AØ
A7
15...7
37
M1
[M1]
+5V
1/2 'LS 74
ARready
18
16
ASstrobe
MREQ
D
S
Q
R
36
IORQ
Z8000 Takt
35
RD
zur Peripherie
Z80A PIO
zu VI Z8000
25
Ø
BØ
B7
27...34
YØ (von 8136)
4
CE
BReady
21
17
DS
23
INT
BStrobe
6
B/A-Sel.
Status 7 VIACK
5
C/D
R/W
11
AØ
A1

Bild 4.64

4.7 ANSCHLUSS-SCHEMATIK FÜR DEN SYSTEM-ZEITGENERATOR-BAUSTEIN AmZ 8073

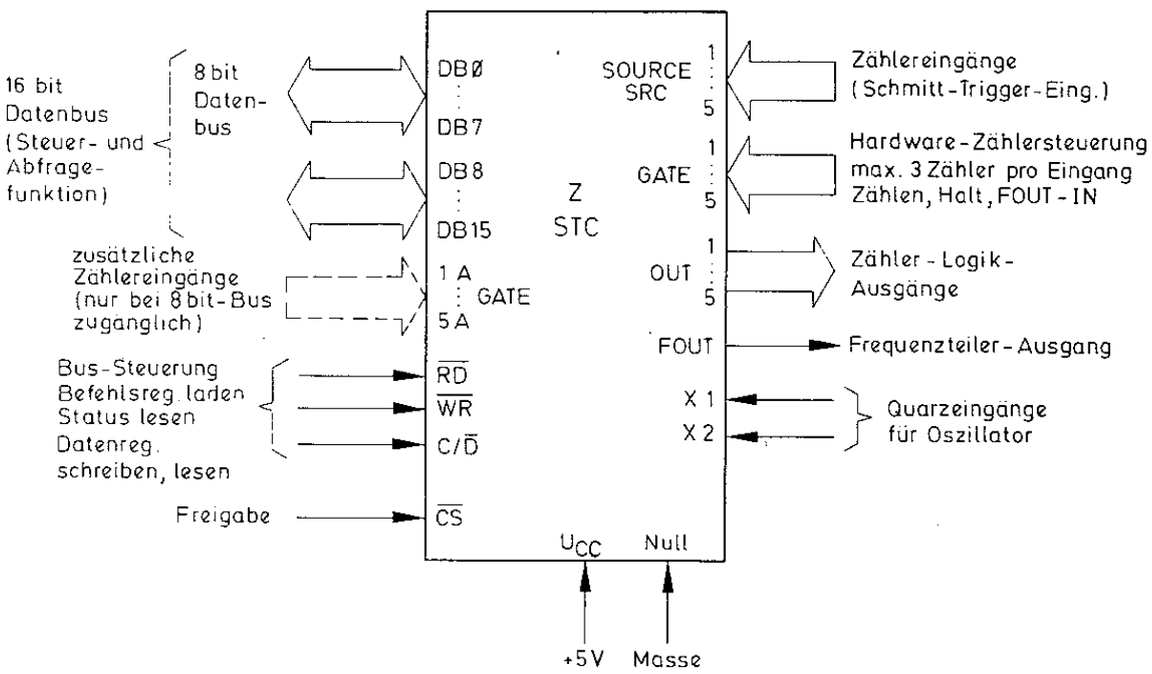
Der Einsatz des Z 8000 als Prozeßrechner wird eine der maßgebenden Anwendungen im breiten Einsatzspektrum dieser CPU bilden. Bei der direkten Prozeßsteuerung spielt die schnelle Echtzeitbearbeitung, eine fein priorisierte Interruptbehandlung und nicht zuletzt eine synchrone Zeitsignalerzeugung eine Rolle. Synchrone Zeitsignale dienen für routinemäßige Auslösungen von Interruptanforderungen (im Gegensatz zu reinen asynchronen Ereignissen, die nicht vorhersehbar sind) als Ersatz softwaremäßig gebildeter Zeitschleifen zu diesem Zweck. Im Grunde dienen Zeitsignale bei Mikrocomputeranwendungen für vielfältige Vorgänge und Operationen, bei denen ein sequentiell koordinierter Ablauf eine große Rolle spielt. Und davon gibt es sehr viele. Es lag demzufolge nahe, einen Baustein zu entwerfen, der Zeitsteuersignale in einer sehr variablen wie auch einfach zu programmierenden Form zu erzeugen vermag. Mit dem Zeitsteuerbaustein AmZ 8073 STC ist dies der Firma AMD in gewünschter Form gelungen. Dieser Mehrfach-Zählerbaustein ist keineswegs nur ein etwas erweiterter Binär- oder BCD-Zähler. Der hoch integrierte LSI-Baustein in NMOS-Technologie (4 Micron Zellstruktur) beinhaltet über 13.000 Transistoren. Damit deutet er auf alles andere als eine einfache interne Struktur der Schaltung hin. Der Am-Baustein wurde (wie das AmZ andeutet), auf eine bewußt direkte Kompatibilität zum Z 8000 hin entwickelt. Der 8073 STC ist über einen 8- oder 16-bit-Datenbus in einer für systembezogene Peripheriebausteine üblichen Initialisierungsroutine programmierbar. Die Integration dieses Bausteins in ein Z 8000-System ist denkbar einfach. Signaltechnisch wird der 8073 STC von der CPU wie ein RAM angesprochen. Das sagt im Prinzip alles.

Bild 4.65 zeigt das Anschlußbild dieses Bausteins. Über die bidirektionalen Anschlüsse DB0 . . . DB15 erfolgt die Verbindung zum Datenbus der CPU. Der AmZ 8073 STC ist ein Zählerbaustein vielseitigster Prägung:

- 5 unabhängig initialisierbare 16-bit-Zähler für Aufwärts- und Abwärtszählbetrieb in binärer oder BCD-Zählart
- Programmierbarer 16-bit-Start- und Stopwert durch jeweilige Lade- und Halteregister für alle 5 Zähler
- Zusätzliche 16-bit-Komparator- und Alarmregister, jeweilig für Zähler 1 und 2, ermöglichen durch Freigabe zweier Bits in einem übergeordneten Betriebssteuerregister die Initialisierung einer echten Tageszeit-Betriebsart (Zähler 2 zeigt Stunden und Minuten, Zähler 1 zeigt Sekunden und Zehntel-Sekunden)
- Quarz-Basisfrequenz von 10 MHz (max.) mit folgender Frequenzskalierung aus 4 x 4-bit-Teilern, programmierbar in Binär- oder BCD-Teilung, als Eingangstaktfrequenz für die fünf Zählergruppen; jede Eingangstaktfrequenz als Quotient von 10 MHz durch 1 . . . 65536 (quantisiert in Einerschritten) ist programmierbar
- Wahlweise Taktversorgung jeder Zählergruppe von der variablen Takterzeugung intern oder einer externen Taktfrequenz möglich; zusätzlich triggerbar an fünf Gate-Eingängen
- Diskretes 1-Bit-Ausgangssignal OUT1 . . . 5 jeder Zählergruppe als Single-Puls, symmetrisches Rechtecksignal oder asymmetrische Pulsform; frei programmierbar Hi- oder Lo aktiv
- Einfache Bussteuerung des Bausteins von der CPU her durch die Eingänge $\overline{RD}$, $\overline{WR}$, $\overline{CS}$ und C/$\overline{D}$ (schreiben/lesen der Steuerregister; laden/lesen der Start- und Halteregister und zusätzlich der Alarm- und Komparatorregister, jeweils einer Zählergruppe).

Diese Auflistung der internen Logik des

Zeitsteuerbaustein
(5 unabhängige 16 bit - Zähler)



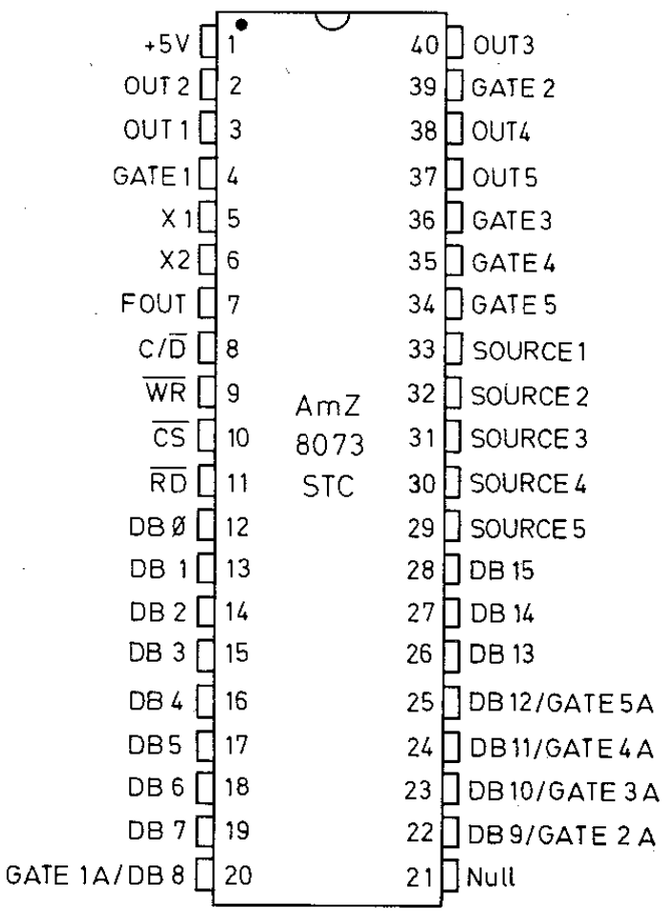


Bild 4.65

8073 STC ist damit noch nicht ganz vollständig, zeigt aber im wesentlichen die Möglichkeiten dieses universellen Zeitgenerator-Bausteins.

Wie angedeutet, enthält der STC zwei Arten von Zählerlogikgruppen. **Bild 4.66** zeigt den Typus von 1 und 2, die ein Komparator- und Alarmregister (beide 16 bit) enthalten. Diese beiden Zählerlogikgruppen beziehen sich auf die Tageszeitbetriebsart. In **Bild 4.67** ist die vereinfachte Zählerlogikgruppe dargestellt, die für 3, 4 und 5 stellvertretend ist. Für alle fünf Gruppen gilt, daß sie eine Eingangssignal-Auswahllogik enthalten, die neben externer Eingänge und Steuerung den internen Takt und alle vier OUT-Leitungen der anderen Zählerlogikgruppen anliegen haben. Damit ist beispielsweise die Hintereinanderschaltung aller fünf Zähler möglich. Man könnte damit Taktzeiten von vielen tausend Jahren programmieren. Die individuelle Zählerlogik-Programmierung wird über ein 16-bit-Zählerbetriebsart-Register vorgegeben. Einzelne Bitgruppen des Registers legen die Taktquelle fest, definieren den aktiven Pegel der Gate-Steuereingänge, definieren die grundsätzliche Zählart einer Logikgruppe (auf, ab,

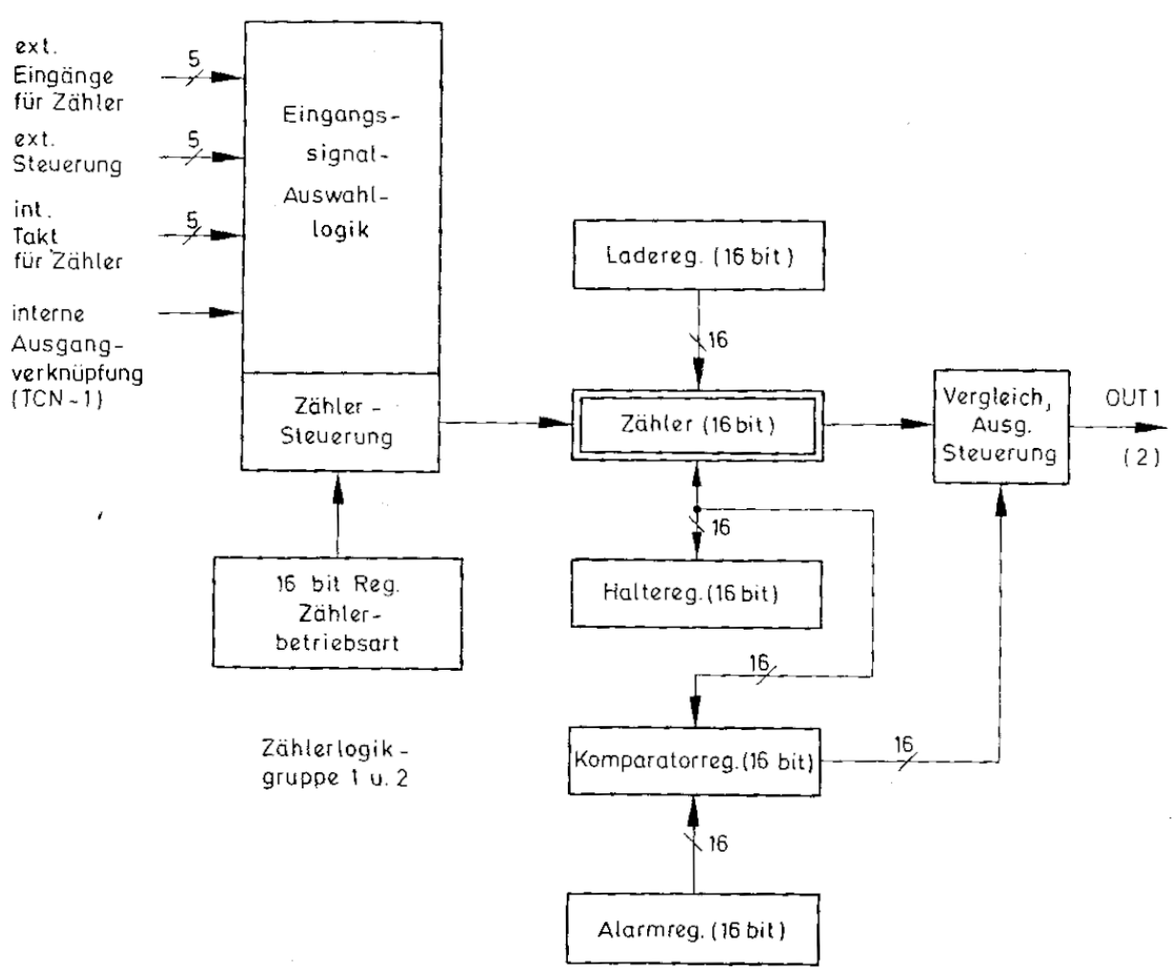


Bild 4.66

binär, BCD, einmalig, repetierend, Ausgangssignaltyp). Die übergeordnete Funktionssteuerung des gesamten Bausteins übernimmt das 16-bit-Master-Mode-Register (MM), in dem all die Funktionen programmiert werden, die für die Zählergruppen gemeinsam gilt. Darunter ist die duale oder BCD-Taktteilung des internen Taktgenerator-Teilers, die Datenbusbreite, die Komparatorregister-Freigabe der Zählerlogiggruppen 1 und 2 und die Programmierung

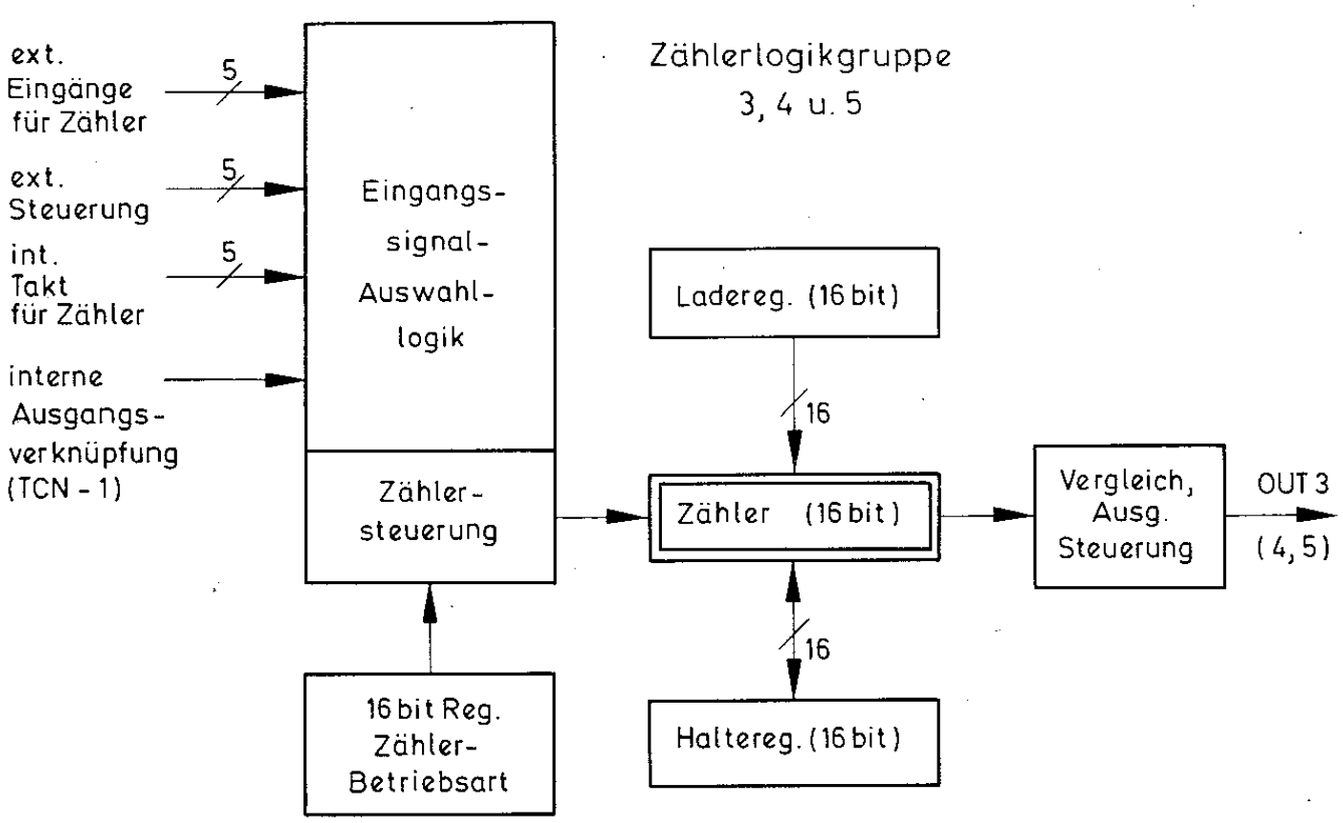


Bild 4.67

der Tageszeitbetriebsart inbegriffen.

In **Bild 4.68** wird eine Interface-Schaltung für einen Anschluß des AmZ 8073 STC zum Z 8000 dargestellt. Die Initialisierung des STC wird vorgenommen, wie normalerweise ein Schnittstellenbaustein spezifiziert wird. Der STC gilt für die CPU als Port, woraus sich ableitet, daß das Signal $\overline{MREQ}$ invertiert auf die Freigabe der Lese- und Schreibsignale $\overline{RD}$ und $\overline{WR}$ an zwei ODER-Gatter gegeben wird. Nur bei einem "Input-/Output"-Befehl kann ein Transfer erfolgen. Ein Adreßdekodierer AmZ 8148 erzeugt aus einem Dekodierausgang ($\overline{Y7}$) die Freigabe des Bausteins an $\overline{CS}$. Die Adreßdekodierung wird, wie üblich aus einer 16-bit-E/A-Adresse gewonnen. Der STC-Eingang C/$\overline{D}$ wird direkt aus der Adreßleitung A1 abgeleitet, worin die Initialisierung des Bausteins bei A1 gleich 1 durchgeführt wird.
Die eigentliche Rückwirkung des STC auf die CPU erfolgt von den Ausgängen OUT1 . . . 5 an den NVI-Interrupteingang (auch NMI verwendbar). Der VI-Eingang (Vektorinterrupt) ist nicht zu verwenden, weil eine spezielle Kennung (Vektor) vom STC nicht produziert werden kann. Über diese Interruptanforderung ist die CPU zu einer wiederkehrenden, synchronen Interruptserviceroutine anzuregen. Mit einer geeigneten Hardware ist eine Anforderung auch in ein Kettenschema einzugliedern.
Die folgende Auflistung der Permutationen der vier Steuereingänge des 8073 STC beschreibt den einzelnen Transfertyp der Operation CPU ↔ STC, der mit dem Interface nach Bild 4.68 ausgeführt werden kann.

CS	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	Datenbus-Operation
Lo	Lo	Lo	Hi	CPU liest die Zähleroperandenregister (Lade-, Halte-, Komparator- und Alarmregister)
Lo	Lo	Hi	Lo	CPU lädt die Zähleroperandenregister mit 16-bit-Werten
Lo	Hi	Lo	Hi	CPU liest die Zählerbetriebsart-Register, das Master-Mode-Register und das 8-bit-Statusregister
Lo	Hi	Hi	Lo	CPU lädt die Zählerbetriebsart-Register und das 8-bit-Datenzeiger-Register
Hi	X	X	X	Datenbus in hochohmigem Zustand

Alle weiteren Permutationen sind nicht erlaubt bzw. treten erst gar nicht auf.

Man kann in einem festen Programm eine Grundinitialisierung vornehmen, der jeweilig den Erfordernissen durch spätere Umprogrammierung Rechnung getragen werden kann. Die Initialisierung ist in zwei Arten durchzuführen: Automatisch nach dem Einschalten und nach einem softwaremäßigen Reset durch den Master-Reset-Befehl; in einzelnen Worten nur zur Änderung, ohne vorausgehenden Reset. Ein softwaremäßiger Master-Reset-Befehl wird an das 8-bit-Datenzeiger-Register durch FFH gegeben. In diesem Datenzeiger-Register des STC sind weitere 16 Befehle mit entsprechenden Bedingungen zu erteilen, die sowohl das Laden des Master-Mode-Registers betreffen als auch das Umladen der Zählerinhalte in dem Halteregister.

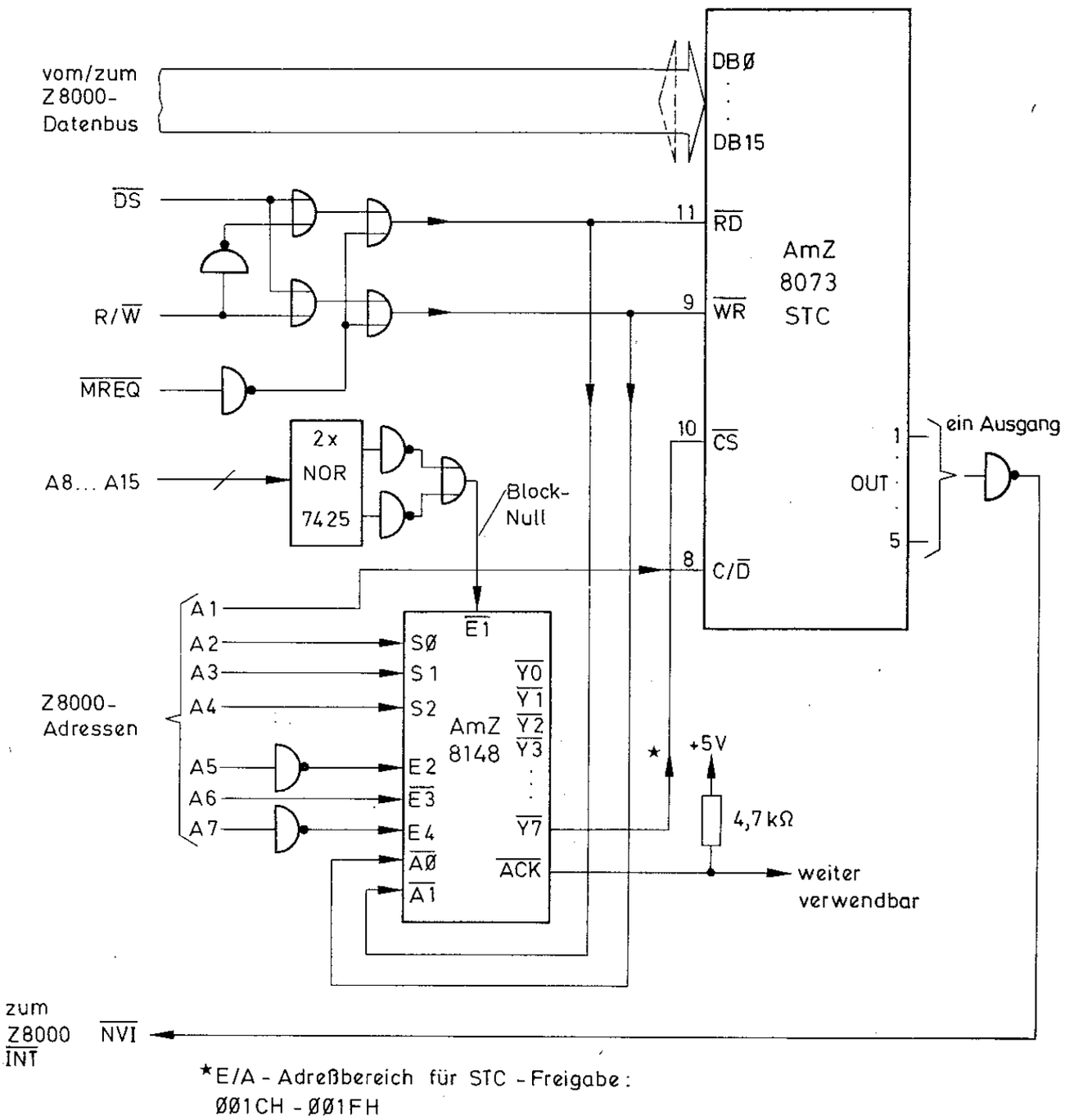


Bild 4.68

Als hardware-technische Anmerkung sei zum Quarzeingang X1, X2 des 8073 STC angemerkt, daß bei einem Quarzbetrieb der Anschluß X2 mit einem Keramiktrimmer von 15 . . . 60 pF gegen Masse zu versehen ist und von X1 47 pF gegen Masse zu legen ist. Soll beispielsweise der STC mit dem CPO-Takt betrieben werden, ist über einen Treiber mit offenem Kollektor (1K-Widerstand auf +5 V) der Takt an X2 anzuschließen. X1 bleibt dabei offen.

Der Amz 8073 STC gilt in der Tat als vielseitig. Anwendungen lassen sich sehr leicht ausdenken und sind dann oft nicht mehr wegzudenken.

4.8 FEHLERKORREKTUR BEIM BETRIEB DYNAMISCHER RAM-SPEICHERBÄNKE

Wird ein größeres Z 8000-System aufgebaut, sind überwiegend dynamische Schreib-/Lesespeicher im Spiel. Das Prädikat "groß" ist wohl sehr relativ. Es sollte damit auch nichts anderes angedeutet werden, als daß mehrere Speicherwerke des Typs verwendet werden, wie sie in Bild 4.36, 4.42 und 4.53 dargestellt sind. Beispielsweise ist ein als groß zu bezeichnendes System mit 20 Daten- und 10 Programmcodesegmenten zu je 64 KB (oder 64K Worte) ausgestattet. Für ein Z 8001-System dieser Größenordnung mit Megabyte-Halbleiter-Speicherbereichen ist wohl die Wahrscheinlichkeit eines Betriebsausfalls eines dynamischen Speicherbausteins oder irgend einer anderen "atmosphärischen" Einwirkung nicht gerade gering, die letztlich einen Fehler an einer Bitposition verursacht. Computer sind von Menschenhand erstellt und durch und durch irdisch. Eine Portion Skepsis gegenüber der Arbeitsfähigkeit eines Computers ist durchaus angebracht. Tritt beispielsweise ein Bit-Fehler an einer potentiell bedeutsamen Stelle auf (z.B. Opcodeteil eines Befehls), kann dies zu Fehlinterpretationen seitens der CPU führen, die ein Programm dann niemals ordnungsgemäß ausführen kann. Sie kann in diesem Fall einen Trap auslösen, was aber nicht sein muß. Ein Trap würde zudem nicht auf die Ursache des Fehlers hinweisen.

Da das Problem von einem (oder mehreren) Bitfehlern dadurch verkompliziert wird, indem der Fehlertypus sehr verschieden sein kann, wird damit schnell klar, daß man bezüglich der Fehlererkennung beim Einsatz großer dynamischer Speicherwerke etwas tun muß. Der Ausfall von großen Computersystemen kostet Geld, was sich sicher von selbst versteht. In einem Computersystem können folgende Fehler auftreten, die im wesentlichen Zusammenhang den Arbeitsspeicher betreffen. Eine Auswahl von vier Fehlern sei hier stellvertretend genannt, die als Einzelfall auf eine einzige Datenleitung bezogen sind.

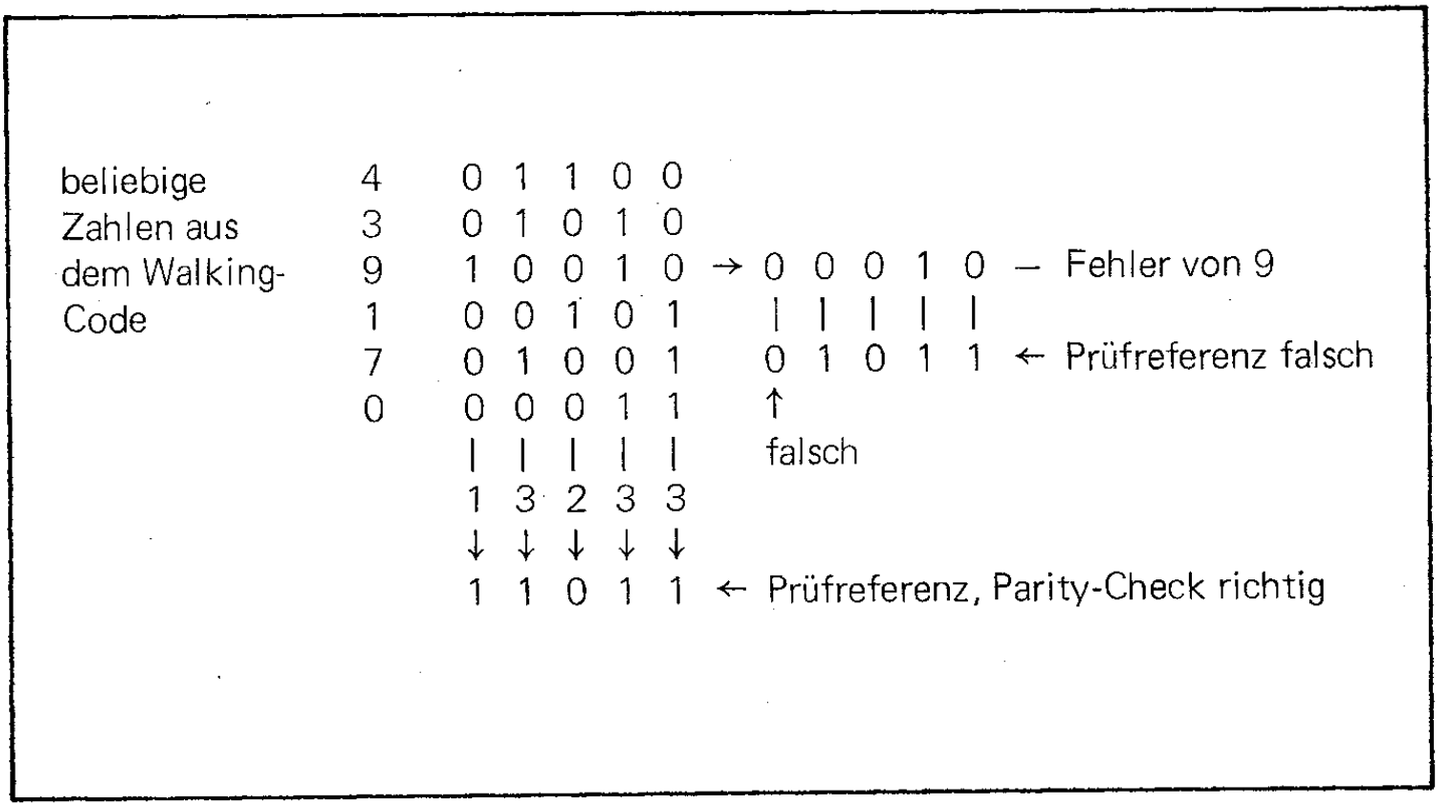
- **Bit ist konstant Hi** → Kurzschluß auf plus; defektes Treibergatter; Speicherbaustein total defekt
- **Bit ist konstant Lo** → Kurzschluß auf Masse; defektes Treibergatter; Speicherbaustein total defekt
- Bit bei einer bestimmten Adresse immer zyklisch Lo (oder Hi) teildefekter Speicherbaustein (Ausfall der Zeilenadreßaktivierung an einer Zeilenleitung)
- Bit bei einer beliebigen Adresse stochastisch Lo (oder Hi) Speicherbaustein höchstwahrscheinlich nicht defekt; Fehler durch Einwirkung elektrischer Spannungsspitzen auf Datenübertragungsleitungen oder durch Einwirkung hochenergetischer α-Strahlung (Teilchen) direkt auf die Speicherbausteine.

Aus diesen vier Punkten läßt sich grundsätzlich der Typus des permanenten Fehlers ableiten, der zumindest dauerhaft genug ist, um ihn rekonstruieren zu können. Dieser Fehler ist rein hardwaretechnischer Art.
Im letzten der vier Punkte ist ein schwer zu lokalisierender Fehlertypus genannt. Er tritt teilweise oder vollkommen willkürlich auf. Die äußere Einwirkung muß keine irreversiblen Schäden hinterlassen (Zerstörung des Bausteins). Jedoch sind unvorhersehbar unkontrollierte Änderungen eines Speicherbits von Lo auf Hi und umgekehrt schwer zu erfassen. Andererseits ist es gar nicht so wichtig, welche Ursachen dieses fehlerhafte Datenbit hatte. Es gilt vielmehr, statistisch zu erfassen, ob bei einem übertragenen Wort eine fehlerhafte Bitposition, zwei fehlerhafte Bitpositionen, drei, vier ... eine angenommene, starke relative Abnahme zeigen. Daraus läßt sich dann ableiten, welchen Aufwand an Stellen man bezüglich des Aufspürens und der Korrektur treiben muß. Eine Untersuchung

von der Firma AMD führte hierzu zu dem Ergebnis, daß bei einer 1000stündigen Betriebszeit eines 16K x 1 dynamischen RAMs (read-modify-write-Betrieb) eine relative Bitfehlerrate von 0,1 % auftritt. Die theoretische Wahrscheinlichkeit eines Doppelbitfehlers in einem übertragenen Datenwort hat sich mit 0,002 % bei 1000 Betriebsstunden herausgestellt. Bei diesem Wert hat man selbstverständlich davon auszugehen, daß permanente Fehler durch Frühausfälle nicht berücksichtigt sind. Es wird eine Fehlerrate angenommen, die im mittleren Bereich der Bausteinbetriebszeit und damit im konstanten Bereich der Ausfallrate liegt. Der genannte praktische Betrieb bezieht sich vielmehr auf das Ereignis willkürlich auftretender Bitfehler, die nicht auf einen defekten Baustein zurückzuführen sind. Und genau mit diesem Fehlertypus hat es ein Aufspür- und Korrekturbaustein vorwiegend zu tun. Das Korrekturerfordernis von Doppelbitfehlern liegt bereits um den Faktor 50 (relativ) hinter dem von Single-Bit-Fehlern, so daß es genügt, die Singles aufzuspüren und zu korrigieren und Doppelbitfehler nur aufzuspüren, anzuzeigen und das Computersystem definiert zu stoppen.

Wie geschieht nun eine Korrektur, bzw. das vorhergehende Aufspüren eines Fehlers. Generell geht man von einem m-aus n-Code aus, der bei einer auftretenden Zahl von Codewörtern aus n Bits (z.B. 16 bit) eine bestimmte Zahl m enthält, deren Stellen mit Einsen belegt sein müssen. Diese Codes sind prüfbar. Als Beispiel seien der pentadische Code (10 aus 32), der Walking-Code, der biquinäre Code oder der Restklassen-Code Wenn man die bei Übertragungsoperationen auftretenden Fehler aufspüren und korrigieren will, muß man auf korrigierbare Codes zurückgreifen. Es gibt einige Korrekturverfahren, von denen die beiden wichtigsten die Codeübertragung im Block und die Anwendung des **Hamming-Codes** ist (R.W. Hamming ist ein amerikanischer Informationstheoretiker).

Bei der Blockübertragung werden mehrere Datenwörter eines prüfbaren Codes zu einem Block zusammengefaßt. Jedem Block wird ein Codewort als Prüfreferenz hinzugefügt. Das Aufspüren von Fehlern erfolgt durch die Paritätsprüfung. Die Anzahl der in den einzelnen Spalten des Blocks auftretenden Einsen werden über einen Parity-Check aus der Prüfreferenz auf eine Zahl ergänzt. Folgendes Beispiel zeigt das:



Wenn die Anzahl der Einsen in einer Spalte gleich 1 oder 3 (ungerade Zahl) sind, muß eine gerade Zahl ergänzt werden, weil damit gleichsam auf eine gerade Zahl ergänzt wurde. Die fehlerhafte Spalte wird angezeigt, indem das neu ermittelte Parity-Wort einen anderen Wert ergibt, als im hinzugefügten Codewort der Prüfreferenz angegeben ist. Der Wert in der entsprechenden Spalte ist anders, siehe aus dem verloren gegangenen Bit der Zahl 9. Um nun eine falsche Zeile ausfindig zu machen, wird jede Zeile auf zwei Einsen hin getestet. Das verlorene Bit des Wertes 9 ergibt eine falsche Quersumme (1 statt zwei). Aus der kombinierten Zeilen- und Spaltenprüfung kann der Fehler des Wertes 9 genau ermittelt und korrigiert werden. In einer Zeile oder Spalte auftretende Mehrfachfehler können nicht mehr korrigiert, aber wenigstens angezeigt werden.

Man benutzt also im Code eine Anzahl Redundanzen zum Erkennen von Fehlern (Redundanz: von 16 möglichen Permutationen werden nur 10 benutzt < tetradischer Code $\Delta H = ld16 - ld10 = 4 - 3{,}32 = 0{,}68$ bit >). Die Korrigierbarkeit auftretender Übertragungsfehler ist durch eine sinnvolle Einführung von Redundanzen zu erreichen. In systematischen Codes werden die Redundanzen in nicht ganz systematischer Weise eingeführt. Wie schon gesagt, haben in einem systematischen Code mit Datenwörtern der konstanten Breite n eine bestimmte Anzahl m immer den Wert 1 (oder 0), d.h. die Quersumme ist immer gleich. Die restliche Zahl von Bits (n – m), die folglich den Wert 0 (oder 1) haben müssen, sind die Prüf- und Felderkennungsbits.

Die Entwicklung eines Fehleraufspür- und Korrekturbausteins muß die Bedingung enthalten, welchen Typus von Fehlern (1 Bit, 2 Bit) man aufspüren und behandeln will. Das hat eine entscheidende Rückwirkung auf den Typ des Codes. Vier Fehlerbehandlungsebenen kann man anpeilen:

- Erkennung eines Einzelbitfehlers
- Korrektur eines Einzelbitfehlers
- Korrektur eines Einzelbitfehlers und Erkennung von Doppelbitfehlern
- Erkennung und Korrektur von Mehrbitfehlern

Um die Forderung der jeweiligen Punkte zu spezifizieren, muß ein weiterer Parameter (der Redundanzfaktor R) genannt werden. Er ergibt sich aus dem Quotient:

$$R = \frac{n}{m}$$

n = Anzahl der Bits des Datenwortes
m = Anzahl gleicher dualer Parität, Quer-Quersumme von Einsen

Die jeweilige Forderung der obigen vier Punkte verlangt nach einem Redundanzfaktor R, der auf ein Minimum optimiert werden muß. Systematische Codes, die diese Forderung erfüllen, sind die korrigierbaren Hamming-Codes.

Hat ein beliebiges Datenwort n Bit Breite und m Information tragende Bits, so sollen die n – m Bits jeweils eine Anzahl der n bit des Datenwortes überprüfen. Dies soll so geschehen, daß die überprüften Bits und das Prüfbit eine gerade Anzahl mit dem Wert Eins (gerade Anzahl Einsen) enthalten. Enthalten die überprüften Bits bereits eine gerade Anzahl von Einsen, wird das zugehörige Prüfbit den Wert Null annehmen. Ist die Anzahl der Einser-Bits ungerade, wird das Prüfbit Eins sein.

An dieser Stelle war ein ausgiebiges Beispiel vorgesehen. Dies wurde bewußt weggelassen, weil es vom Thema Z 8000 etwas zu weit wegführt. Was an dieser Stelle gezeigt werden sollte, ist die interne Arbeitsweise des EDC-Bausteins AmZ 8160 (**e**rror **d**etection and **c**orrection unit). Die Funktion dieses extremen AMD-Bausteins ist nicht gerade "leicht verdaulich". Deshalb wir die Erklärung über diesen Baustein auf das Wesentliche beschränkt.

Der AmZ 8160 EDC arbeitet zur Überprüfung (check) eines 16-bit-Datenwortes mit 6 Prüfbits, gemäß eines leicht

modifizierten Hamming-Codes. Über eine entsprechend aus XOR- und XNOR-Gattern bestehende, interne Prüfbitkodierung werden aus 8 Datenbits die Prüfbits ausgewählt. Für jedes Datenwort wird eine Checknummer ermittelt, die in Bezug auf die überprüften Stellen im Datenwort unmittelbar auf eine falsche Bitübertragung hindeutet. Die Prüfbits sind gemäß der internen Prüfbitkodierung auf das Datenwort projeziert. Folgende Tabelle zeigt die Prüfbitkodierung nach dem modifizierten 16-bit-Hamming-Code:

Erzeugte Prüfbits	Parität	Betroffene Datenbits															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CX	gerade XOR		x	x	x		x			x	x		x			x	
C0	gerade XOR	x	x	x		x		x		x		x		x			
C1	ungerade XNOR	x			x	x			x		x	x			x		x
C2	ungerade XNOR	x	x				x	x	x				x	x	x		
C4	gerade XOR			x	x	x	x	x	x							x	x
C8	gerade XOR									x	x	x	x	x	x	x	x

Die Prüfbits lauten CX, C0, C1, C2, C4 und C8. Das Prüfbit CX wird wahlweise Null oder Eins, je nachdem, ob sich eine gerade oder ungerade Zahl von Einser-Bits im Datenwort bezüglich der Prüfbits ergab. Der AmZ 8160 EDC kann auf 32 bzw. 64 bit kaskadiert werden, wobei sich eine entsprechende Prüfbit-Tabelle, wie die gezeigte, ergibt, die auf 32- bzw. 64-bit-Datenworte projeziert wird. Dies sollte hier nur erwähnt sein, da es ja für den Z 8000 nicht zutrifft.

Aus dem 16-bit-Datenwort wird ein neuer Satz von Prüfbits ermittelt und mit dem übertragenen Prüfbit-Code vom Speicher verglichen. Ist eine Übereinstimmung erzielt worden, ist kein Fehler im übertragenen Datenwort aufgetreten (siehe Beispiel der 6 Datenworte nach dem Walking-Code).

Der AmZ 8160 ist in der Lage, Single-Bit-Fehler aufzuspüren und zu korrigieren, alle Doppelbitfehler aufzuspüren und anzuzeigen und sogar einzelne Dreibitfehler. Grundsätzlich entdeckt er die permanente Fehlerbedingung "alles 0" und "alles 1". Er kann permanente Hardwarefehler allgemein und kritische, stochastisch auftretende Fehler aufspüren und korrigieren. Absolut extrem ist seine Schnelligkeit. Er kann Einzelbitfehler in 50 ns korrigieren. Das ist immens schnell.

4.8.1 Beispiel eines dynamischen RAM-Speichersystems mit dem EDC 8160

Bild 4.69 zeigt das Anschlußbild dieses 48poligen AMD-Monstrums, der in bipolarer Technologie entworfen wurde. **Bild 4.70** zeigt den Baustein und seine Anschlüsse in einer etwas transparenteren Art. Über den Datenbus D0 . . . D15 werden die Datenworte empfangen, die eigentlich die CPU angefordert hat. Dieser Umweg über den 8160 EDC ist notwendig, denn in den vom Speicherwerk ausgegebenen Worten befinden sich die Fehler, die man korrigieren will. Ebenso wird über die Datenanschlüsse das korrigierte Wort (bei Einzelbitfehlern) wieder auf den CPU-Datenbus zurückgegeben (in dem bereits genannten Zeitraum von 50 ns), so daß die CPU. u.U. keinen Wartezyklus einfügen muß. Voraussetzung dabei ist die Verwendung von Speicherbausteinen mit einer entsprechend kurzen Zugriffszeit (≈ 150 ns). An den Eingängen CB0 . . . CB6 werden dem 8160 EDC aus einem separaten Speicher die Prüfbit-Permutation (7 bit) übergeben, nach denen er den Vergleich zu seinen selbst ermittelten Prüfbit-Permutationen anstellt. Ein wichtiger Eingang des EDC ist $\overline{\text{GENERATE}}$. Wird er auf Lo gelegt, erzeugt er intern selbst ein Prüfbitmuster, das er an den "Syndrom"-Ausgängen SC0 . . . SC6 ausgibt. Wird dieser Eingang auf Hi gelegt, arbeitet der 8160 EDC nach dem externen Prüfbitmuster im Erkennungs-und (nach Wunsch) Korrekturmodus. An einem weiteren Eingang mit der Bezeichnung Korrektur kann man extern definieren, ob der EDC ein übernommenes Datenwort korrigieren soll (Korrektur gleich Hi) oder nicht (Korrektur gleich Lo). Bei Lo werden Fehler nur aufgespürt und angezeigt. Das Datenwort wird vom Eingangszwischenspeicher direkt in den Ausgangszwischenspeicher übergeben.

Entscheidend sind die EDC-Ausgänge $\overline{\text{ERROR}}$ und $\overline{\text{MULTI-ERROR}}$, an denen das Aufspüren der Fehler präsentiert wird. $\overline{\text{ERROR}}$ wird auf Lo springen, wenn ein oder mehrere Fehler entdeckt wurden. $\overline{\text{MULTI-ERROR}}$ wird auf Lo springen, wenn grundsätzlich mehr als ein Fehler im Datenwort erkannt wurde. Sind die Ausgänge Hi, ist kein Fehler entdeckt worden. Diese Ausgänge sind entweder als Interrupt-Anforderungsleitungen an die CPU zu benutzen oder in einen Zähler zu geben, der bei einem Testlauf des Speicherwerks über einen bestimmten Zeitraum die Anzahl der "Syndrome" dokumentiert. Jedes Lo war ein Fehler (oder mehrere) in einem Datenwort. Man kann sehr einfach die Einzelbitfehler dokumentieren, indem man die Ausgänge $\overline{\text{ERROR}}$ und $\overline{\text{MULTI-ERROR}}$ mit einem UND-Gatter verknüpft und auf einen separaten Zähler gibt. Jedes Lo inkrementiert den Zähler und sorgt für eine exakte Zahl. Damit läßt sich zwischen Ein- und Mehrbitfehlern unterscheiden.
Als wichtig seien noch die Code-Definitionseingänge ID0 . . . ID2 genannt. Für ein 16-bit-Speicherwerk des Z 8000 ist praktisch nur die Permutation ID0 = ID1 = ID2 = Loswichtig. Sie definiert den Hamming-Code für 16 Datenbits und 6 Prüfbits, wie sie in der Tabelle in 4.8 angegeben wurden. Eventuell ist noch der Eigenprüfmodus des EDC mit der Permutation ID1 =ID2 = Lo und ID0 = Hi wichtig. Alle anderen Permutationen an den ID-Eingängen betreffen 32- und 64-bit-Speicherwerke, die eine Kaskadierung des 8160 EDC erfordern.

Mit einer Hardwareerweiterung neben dem EDC-Baustein und dem Interface zur Datenumschaltung muß man allerdings leben. Wie in dem Beispiel bei der Codeübertragung im Block dargestellt wurde, bei der man von einer bestimmten Redundanz ausgegangen ist, ist für ein generelles 16-bit-Datenspeicherwerk nicht auszugehen. Die Quersumme jedes Datenwortes hat keinesfalls einen konstanten Wert. Die Gegebenheit eines Codes liegt nicht vor und wäre partiell rein zufällig. Die Tatsache, daß die Datenworte beliebig binär verknüpft sind,

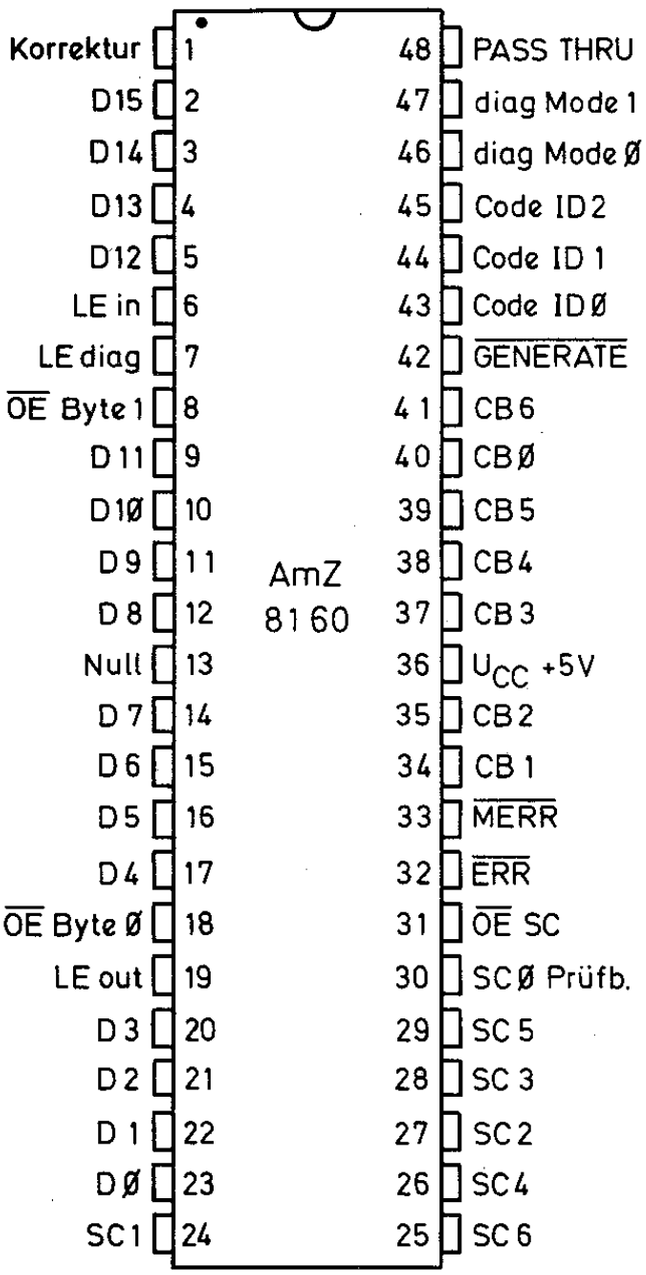

Korrektur 1
D15 2
D14 3
D13 4
D12 5
LE in 6
LE diag 7
OE Byte 1 8
D 11 9
D 10 10
D 9 11
D 8 12
Null 13
D 7 14
D 6 15
D 5 16
D 4 17
OE Byte Ø 18
LE out 19
D 3 20
D 2 21
D 1 22
D Ø 23
SC 1 24
AmZ
81 60
48 PASS THRU
47 diag Mode 1
46 diag Mode Ø
45 Code ID 2
44 Code ID 1
43 Code ID Ø
42 GENERATE
41 CB 6
40 CB Ø
39 CB 5
38 CB 4
37 CB 3
36 UCC +5V
35 CB 2
34 CB 1
33 MERR
32 ERR
31 OE SC
30 SC Ø Prüfb.
29 SC 5
28 SC 3
27 SC 2
26 SC 4
25 SC 6

Bild 4.69

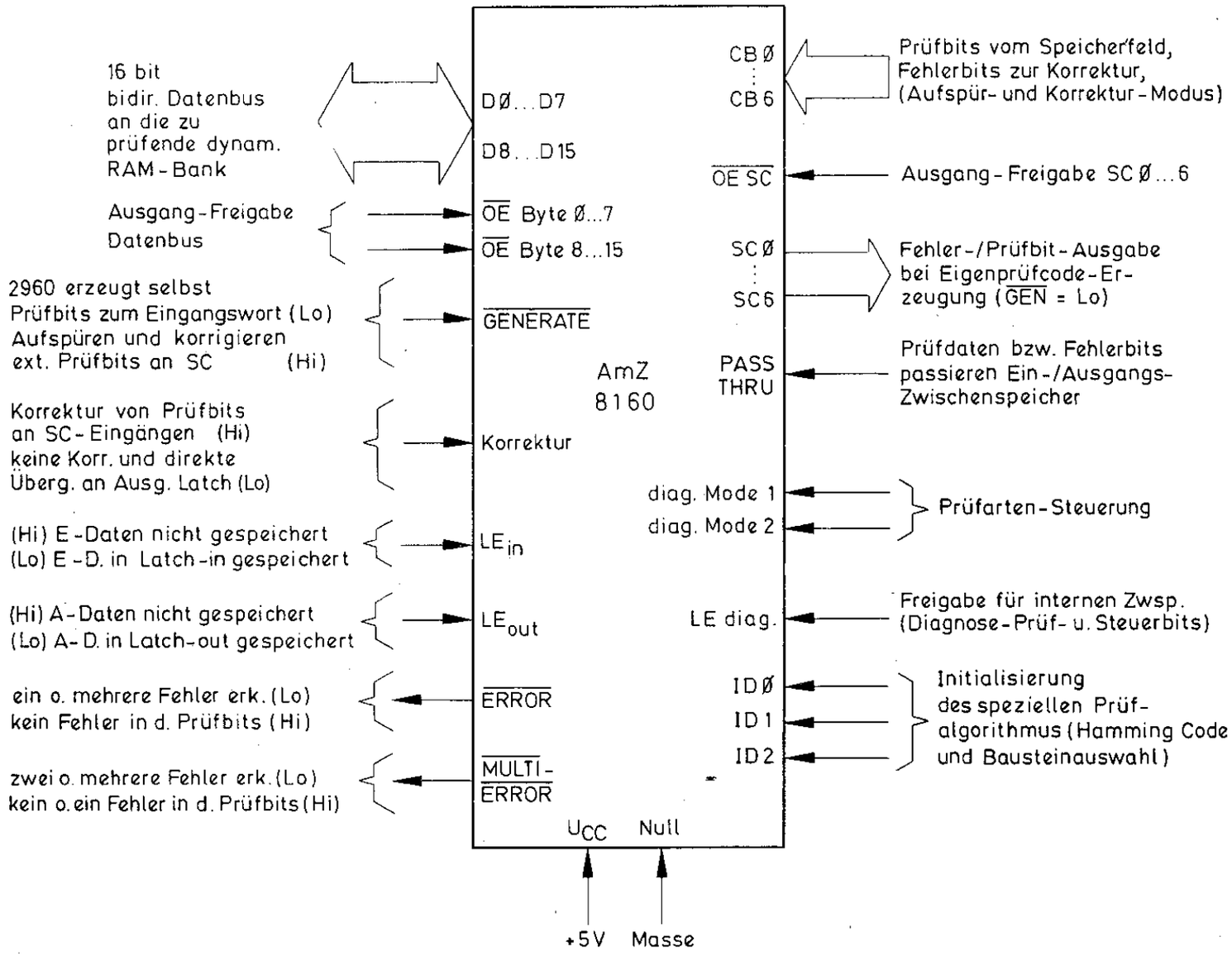

16 bit
bidir. Datenbus
an die zu
prüfende dynam.
RAM-Bank
Ausgang-Freigabe
Datenbus
2960 erzeugt selbst
Prüfbits zum Eingangswort (Lo)
Aufspüren und korrigieren
ext. Prüfbits an SC (Hi)
Korrektur von Prüfbits
an SC-Eingängen (Hi)
keine Korr. und direkte
Überg. an Ausg. Latch (Lo)
(Hi) E-Daten nicht gespeichert
(Lo) E-D. in Latch-in gespeichert
(Hi) A-Daten nicht gespeichert
(Lo) A-D. in Latch-out gespeichert
ein o. mehrere Fehler erk. (Lo)
kein Fehler in d. Prüfbits (Hi)
zwei o. mehrere Fehler erk. (Lo)
kein o. ein Fehler in d. Prüfbits (Hi)
DØ...D7
D8...D15
OE Byte Ø...7
OE Byte 8...15
GENERATE
Korrektur
LEin
LEout
ERROR
MULTI-ERROR
AmZ
8160
UCC
Null
+5V
Masse
CBØ
CB6
OE SC
SCØ
SC6
PASS THRU
diag. Mode 1
diag. Mode 2
LE diag.
IDØ
ID1
ID2
Prüfbits vom Speicherfeld,
Fehlerbits zur Korrektur,
(Aufspür- und Korrektur-Modus)
Ausgang-Freigabe SCØ...6
Fehler-/Prüfbit-Ausgabe
bei Eigenprüfcode-Er-
zeugung (GEN = Lo)
Prüfdaten bzw. Fehlerbits
passieren Ein-/Ausgangs-
Zwischenspeicher
Prüfarten-Steuerung
Freigabe für internen Zwsp.
(Diagnose-Prüf- u. Steuerbits)
Initialisierung
des speziellen Prüf-
algorithmus (Hamming Code
und Bausteinauswahl)

Bild 4.70

hat für den Redundanzfaktor zur Folge, daß er immer unterschiedlichen Wert hat. Die Prüfbit-Permutation gilt nur für ein einziges Datenwort an einer Speicheradresse. Unter der Hardwareerweiterung ist also die Erweiterung des Speicherwerks gemeint. Das bedeutet, daß für ein 32K-Worte-Speicherwerk ein 32K x 6-bit-Prüfbitmuster-Speicherwerk integriert werden muß. Jedes 16-bit-Datenwort bekommt an der gleichen Adresse ein 6 bit breites Prüfbitmuster zur Seite, das beim Beschreiben der Adresse an den separaten 6-bit-Speicherplatz abgelegt werden muß. Beim Lesen wird dieser Speicherplatz mit ausgelesen und vom EDC an den CB-Eingängen übernommen. In **Bild 4.71** schließlich ist ein Speicherwerk dargestellt (bestehend aus dynamischen RAMs), das eine gesamte Wortlänge von 22 bit beinhaltet (16-bit-Daten, 6-bit-Prüfcode). Den Schnittpunkt zwischen Daten zum und vom Speicherwerk, zum und vom EDC und letztlich zum und vom Z 8000-Systemdatenbus wird von 4 Bustreiber-Multiplexern AmZ 8162 (nicht invertierender Baustein) gebildet. Diese Bausteine bilden gewissermaßen das Zentrum eines Dreiecks der Datenbuspfade. Beim Lesen des Speicherwerks wird zuerst der EDC bedient. Liefert er das Datenwort wieder zurück, schalten die Multiplexer zum Systemdatenbus um. Dies erfolgt selbstverständlich nur bei einem Lesevorgang der CPU. Will die CPU in den Speicher schreiben, ist der Datenpfad wie gehabt, nämlich direkt rein in das Speicherwerk. Diese genannten Umschaltvorgänge auszulösen obliegt der Bus-Steuer-Interface-Logik, die sowohl den EDC als auch den Bus-Multiplexer bedient. Die Auslösung erfolgt von den Signalen $\overline{AS}$, $\overline{DS}$, B/$\overline{W}$, R/$\overline{W}$ und $\overline{MREQ}$.

Auch diese Logik läßt sich am einfachsten wieder durch die schon einmal genannten PAL-Bausteine entwerfen (programmierbare Gatterlogik), weil die Erstellung der Schaltbedingungen mit Hilfe eines PAL-Assemblers sehr leicht in Baustein-Programmieranweisungen umzusetzen ist. Es müssen nur die definierten Ausgangsbedingungen den gegebenen Eingangsbedingungen zugeordnet werden, wie es für eine PAL-Anwendung in Bild 4.44 exemplarisch gezeigt wurde. Die Zuordnung muß man noch im Kopf machen.

Abschließend sei zu Abschnitt 4.8 gesagt, daß die Fehlerkorrektur als kompliziert zu betrachten ist. Mit den bestehenden AMD-Bausteinen ist sie jedoch überschaubar lösbar geworden. Daß die Fehlerkorrektur immer mehr an Bedeutung gewinnt, kann keine Zweifel hegen. Hochleistungs-Computer mit kapitalem kommerziellen Einsatz müssen noch zuverlässiger werden. Die Mikroelektronik muß ihre Mittel und Wege aktivieren, genau dann zur Korrektur ihrer selbst zu gelangen, wenn der leichte Weg der mehrfach redundanten Systeme zu teuer wird.

4.9 TOPOLOGIE UND KONZEPTION EINES KLEINEN Z 8002-SYSTEMS

In den bisherigen Abschnitten über die Hardware zur Z 8000-CPU wurden bereits vielfältig Anregungen zu Systemen verschiedener Zielsetzung gegeben. Daß man mit sehr einfachen Mitteln bereits ein interessantes Z 8000-System aufbauen kann, soll in diesem Abschnitt gezeigt werden. Man könnte in dem vorgestellten System einen gewissen Bastlerzuschnitt sehen. Dies sollte auch nicht verleugnet werden. Jedoch ist derjenige engagierte Hardwareinteressent mit ernsthafter Blickrichtung auf den Z 8000 keinesfalls mehr ein Bastler im üblichen Sprachgebrauch. 16-bit-Interessenten qualifizieren sich nicht absolut als Weizen von der Spreu, bekunden aber doch mit ihrer Entscheidung eine Absicht auf Anwendungen, die über triviale Spielereien hinausgeht, wenn auch die CPU

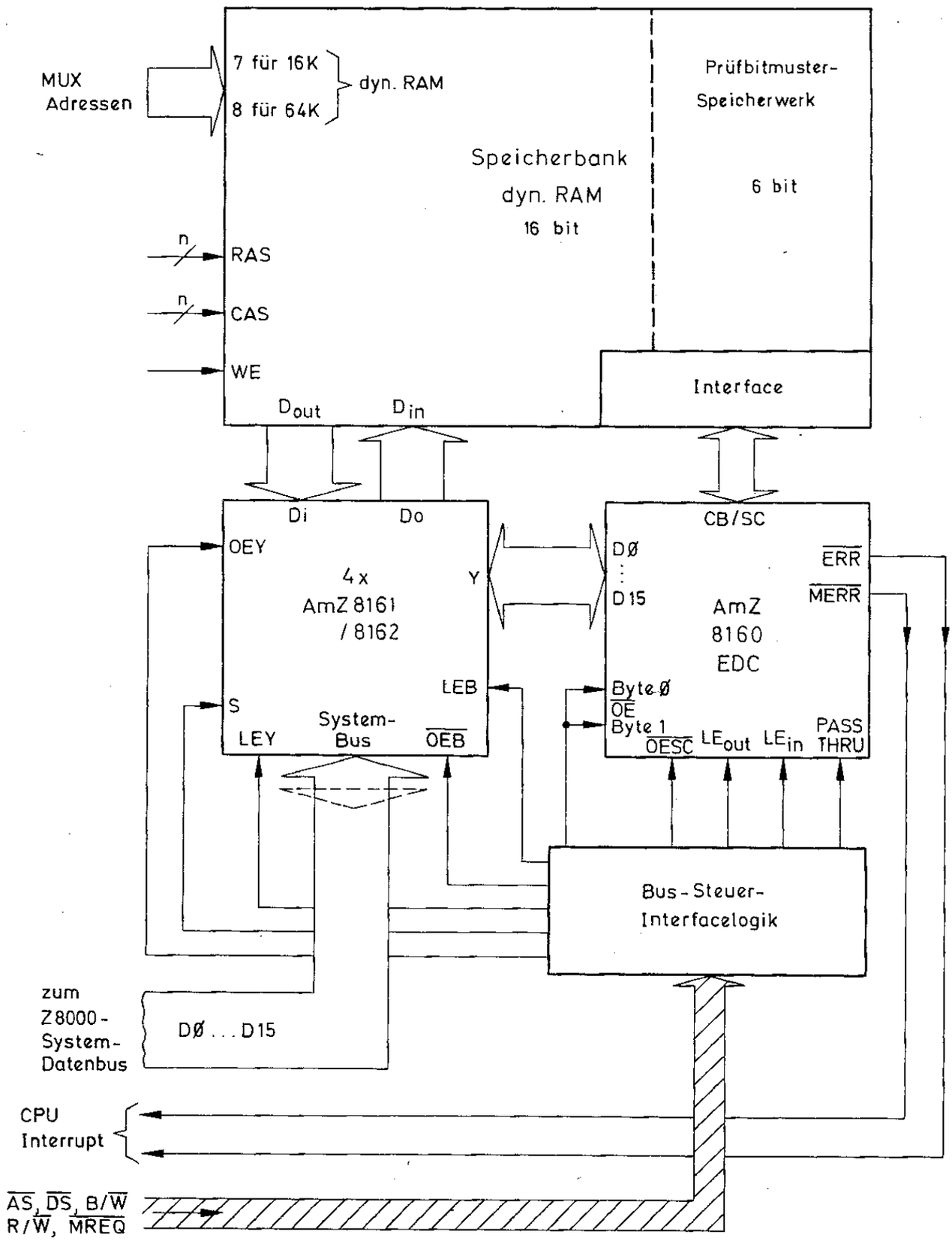

MUX
Adressen
7 für 16K
8 für 64K
dyn. RAM
Speicherbank
dyn. RAM
16 bit
Prüfbitmuster-
Speicherwerk
6 bit
n
RAS
n
CAS
WE
Interface
Dout
Din
Di
Do
CB/SC
OEY
4x
AmZ 8161
/ 8162
Y
DØ
D15
ERR
MERR
AmZ
8160
EDC
LEB
S
LEY
System-
Bus
OEB
Byte Ø
OE
Byte 1
OESC
LEout
LEin
PASS
THRU
Bus-Steuer-
Interfacelogik
zum
Z8000-
System-
Datenbus
DØ . . . D15
CPU
Interrupt
AS, DS, B/W
R/W, MREQ

Bild 4.71

sehr zu Softwarespielereien einlädt. Im Grunde sollte mit dem Hardwaresystem in **Bild 4.72** nur angedeutet sein, daß dieses System mit finanziell leicht vertretbaren Mitteln realisiert werden kann.

Rein hardware-technisch gibt es zu dem kleinen System nicht allzuviel zu sagen, weil es eine konsequente Anwendung von Grundschaltungen ist. Das System besteht aus 8K Worte RAM, 4K Worte Festwertspeicher, einem parallelen Port und einem seriellen Port, an den auf der externen Seite ein Modem oder ein RS-232C-Interface geschaltet werden kann. Die eingezeichnete Verkettung beider Ports mit IEI – IEO muß nicht sein. Es liegt im Ermessen des Anwenders und seinen Erfordernissen. Die Interruptausgänge (Anforderungsleitungen) sind UND zu verknüpfen, wenn es keine offenen Kollektorausgänge sind. Für den gezeichneten Fall wäre als Serienschnittstelle eine Z 80 SIO/O oder ein Z 80 DART (zweifach voll Duplexschnittstelle) einzusetzen. Als Parallelport ist die Z 80 PIO zu verwenden. Es soll hier darauf hingewiesen werden, daß man für ihren Anschluß die Hinweise in Abschnitt 4.6 beachten muß. Das gilt für die SIO (oder DART) wie für die PIO. Wird kein verketteter, priorisierter Interruptbus benötigt, erfüllt der Am 2950 als paralleler und der altbekannte Am 9551 (8251) als serieller Port vollkommen seinen Zweck.

In der Schaltung wurde bewußt genauer auf die einfachste Art der E/A-Adreßseparierung Wert gelegt, weil man diese paar Adreßwerte zur Freigabe des jeweiligen Ports benötigt. Hier wurden mit drei 8-bit-Vergleichern drei frei wählbare Adressen auf der sogenannten "Zero-Page" generiert. Obwohl der Z 8000 kein üblicher seiten-orientierter Prozessor ist, kann diese Seitenbezeichnung mit 256 Byte-Bünden quasi so ausgedrückt werden. Auf die Zero-Page gelangt man durch das Ausblenden aller Adreßwerte über 00FFH mit Hilfe des NOR-Gatters 7425. Das untere Adreßbyte wird verglichen und an $\overline{E_{out}}$ eine Koinzidenz auf ein ODER-Gatter gegeben, das mit dem CPU-Status Nr. 2 (E/A allgemein) eine effektive E/A-Adresse separiert. Mit dieser Leitung wird der Freigabeeingang $\overline{CS}$ der Ports angesteuert bzw. auch C/$\overline{D}$.
Ein weiteres Augenmerk soll auf den Startmonitor des kleinen Systems gelenkt werden. Das kleine PROM mit 32 Worten (zwei Bausteine) hat den Zweck einer definierten Initialisierung nach dem Einschalten oder nach einem System-Reset. Der Z 8002 holt sich automatisch nach einem dieser Ereignisse erst einmal seinen FCW-Wert (Flag- und Steuerwort) an 0002H und an 0004H den Programmzähler-Startwert. Das sind vier Bytes, die an diesen Stellen unbedingt definiert sein müssen. Die PROM-Plätze können überdies mit einer zusätzlichen Initialisierungsroutine belegt werden. Wird der Programmzähler an 0004H mit 0006H geladen, ließe sich an dieser Stelle gleich damit beginnen. Betroffen sind die Werte für das Refreshregister, den Programmstatus-Bereichszeiger und den Normal-Stapelzeiger. Für das bereits geladene FCW-Register besteht die Bedingung, daß Bit Nr. 14 mit der Bezeichnung S/$\overline{N}$ (System/Normal-Modus) unbedingt auf 1 steht, damit das kleine Initialisierungsprogramm im System-Modus ablaufen kann. Die benötigten LDCTL-Befehle sind privilegiert und werden nur im System-Modus ausgeführt. Das kleine Initialisierungsprogramm kann folgendermaßen aussehen:

```
        LD     R0, %0008 (REFRESH)
        LDL  RR2, %000C (PSAP, NSP)
LDCTL REFRESH, R
LDCTL      PSAP, R2
LDCTL       NSP, R3
        JP          %F000 (Hauptmonitor)
```

Dieses kleine Programm paßt leicht in das Startmonitor-PROM. Der bedingungslose Sprung am Schluß lädt den Programmzähler mit der Startadresse des eigentlichen Hauptmonitors, der mit einer Eingaberoutine beginnt. Es wird ein Port abgefragt, an dem die Tastatur für den Bediener angeschlossen ist. Ist an dem seriellen Port eine RS-232C-Schnittstelle angebracht, kann ein belie-

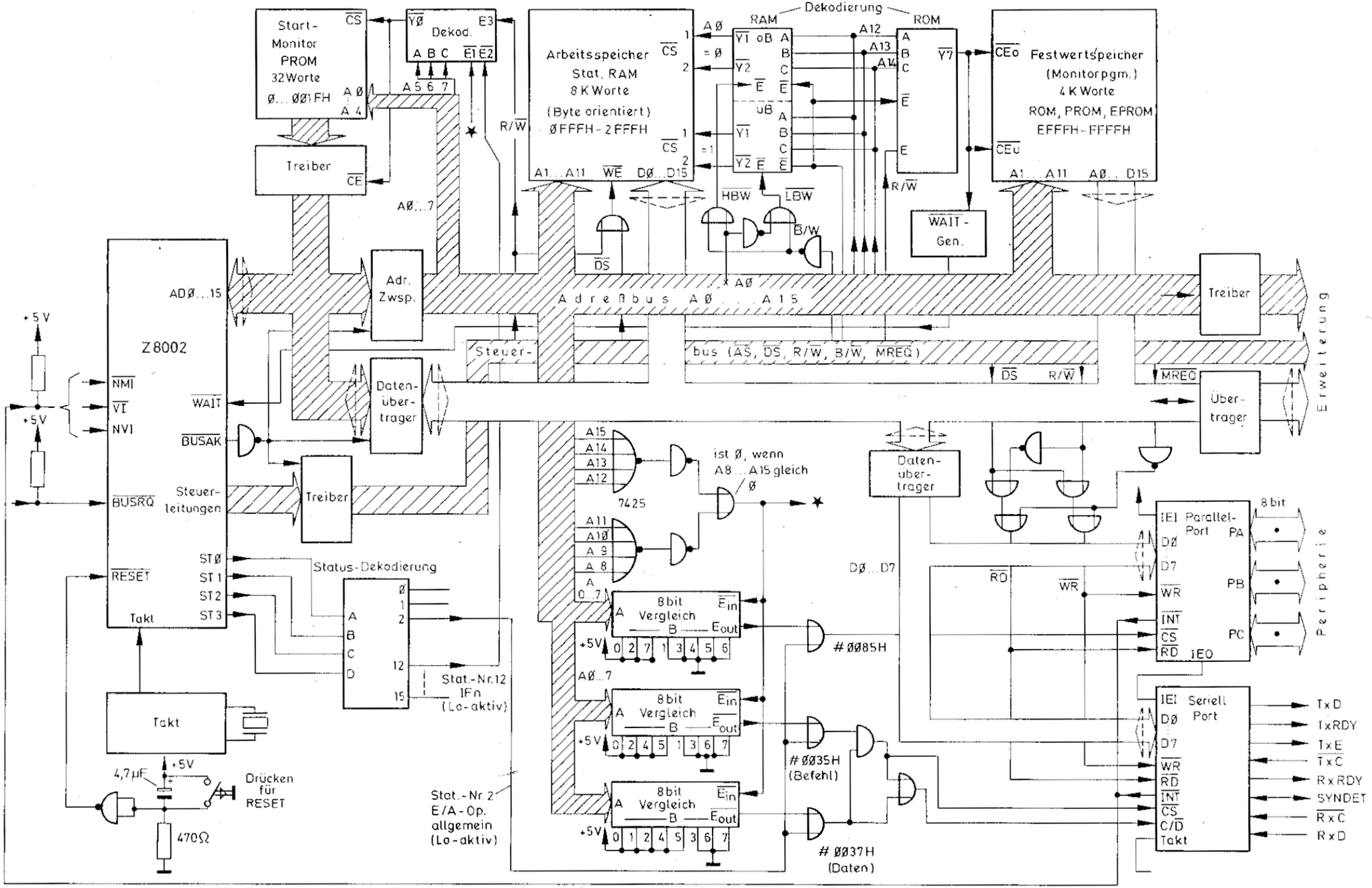
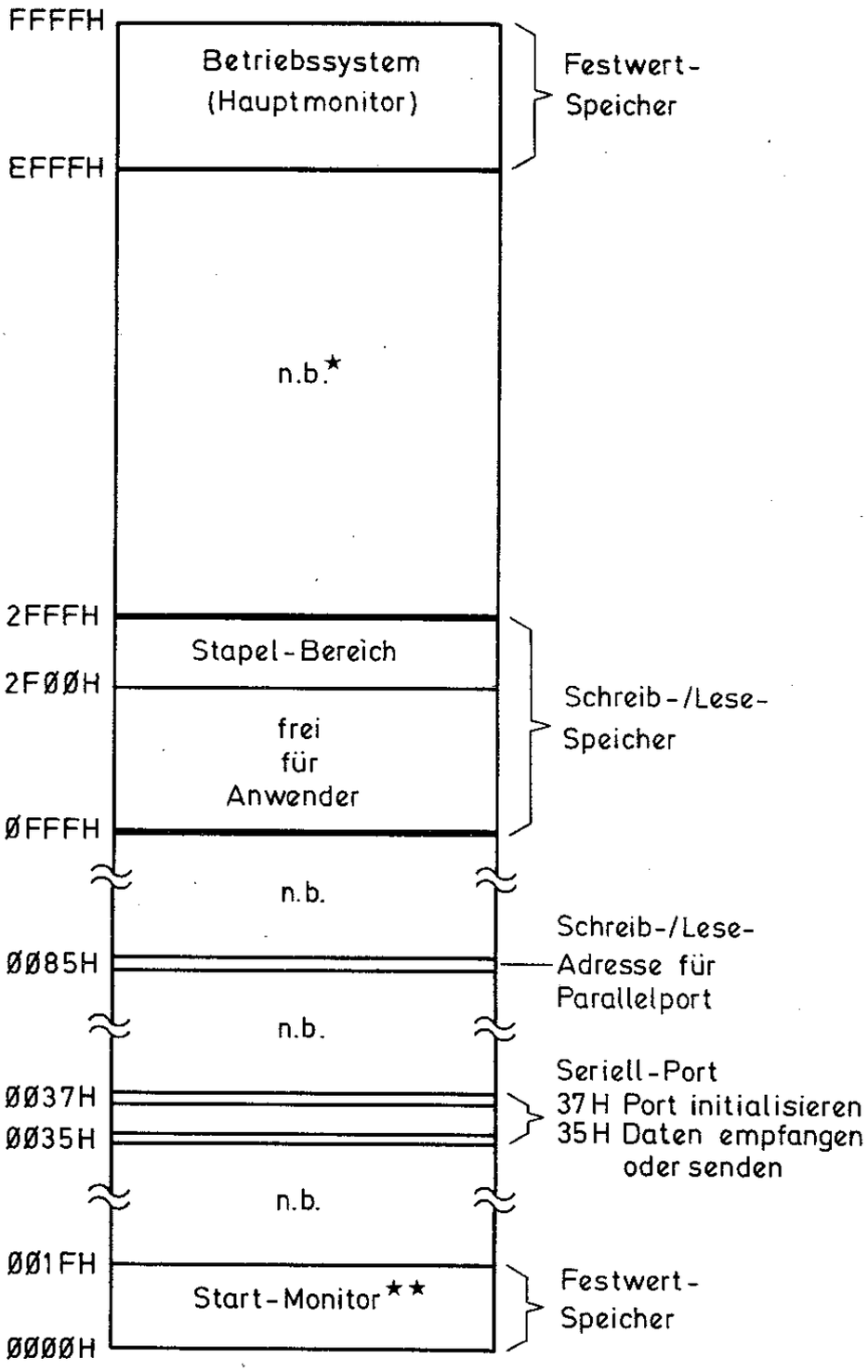

Start-
Monitor
PROM
32 Worte
Ø...ØØ1FH
Dekod.
Treiber
Arbeitsspeicher
Stat. RAM
8 K Worte
(Byte orientiert)
ØFFFH-2FFFH
RAM
ROM
Dekodierung
Festwertspeicher
(Monitorpgm.)
4 K Worte
ROM, PROM, EPROM
EFFFH-FFFFH
WAIT-
Gen.
Adr.
Zwsp.
Steuer-
Daten-
über-
trager
Treiber
Z 8002
Steuer-
leitungen
Takt
Status-Dekodierung
Stat.-Nr.12
IEn
(Lo-aktiv)
Stat.-Nr.2
E/A-Op.
allgemein
(Lo-aktiv)
7425
ist Ø, wenn
A8 ...A15 gleich
Ø
8bit
Vergleich
#ØØ85H
#ØØ35H
(Befehl)
#ØØ37H
(Daten)
Daten-
über-
trager
Über-
trager
Erweiterung
Peripherie
8 bit
Parallel-
Port
Seriell
Port
TxD
TxRDY
TxE
TxC
RxRDY
SYNDET
RxC
RxD
4,7µF
470Ω
Drücken
für
RESET
Takt

Bild 4.72

Adreßfeld für das Z8002-
Kleinsystem



*nicht benutzt, frei für Erweiterungen
**muß in diesem Adreßbereich sein

Bild 4.73

biges Terminal angeschlossen werden, das über einen Anschluß dieses Formats verfügt. Über "Return" wird die Routine gestartet, die einen Wert oder den Inhalt eines Zeilenpuffers in den RAM ab einer gewählten Startadresse beginnend ablegt. Das Prinzip solcher Routinen ist hinreichend bekannt.

Zu dem Refresh-Wert sei gesagt, daß bei der Verwendung eines statischen RAM-Werks das Bit 15 des Wortes Null sein soll (z.B. 00XXH). Das bewirkt generell die Ausschaltung der Erzeugung CPU eigener Refreshzyklen. In **Bild 4.73** ist die Adreßfeldeinteilung für das kleine System dargestellt. Den Wert für den Programmstatus-Bereichszeiger und den Normal-Stapelzeiger muß man sich entsprechend im Feld des gegebenen RAM-Bereichs einteilen. Das ist Eigensache (z.B. 2FFFH).

Zu der Freigabedekodierung des Startmonitor-PROMs am Eingang $\overline{E2}$ des Dekodierers sei bemerkt, daß der Bedingungseingang statisch auf Lo zu legen ist, wenn die Initialisierungsroutine im PROM untergebracht werden soll. Dies ist notwendig, weil der Status IFn Nr. 12 nicht ständig aktiv ist. Er ist nur zwischen 0002H . . . 0005H aktiv.

In der gezeichneten Form bietet das System auf einer Doppel-Europaplatine Platz. Um von vorneherein für RAM-Erweiterungen die Voraussetzung zu schaffen, sollte der Adreß- und Datenbus mit entsprechenden Treibern (2 mal AmZ 8144) für den Adreßbus und Übertragern (2 mal AmZ 8104) für den Datenbus versehen werden.

Der Speicherorganisations-Baustein Z8010MMU, ein Kernbaustein des Z8000-Systems

5

Die Zielsetzung beim Entwurf eines Z 8000-Systems liegt in einer höheren Ordnung variabler Anwendungen. Betrachtet man ein echtes Hostcomputersystem mit dem Einsatzziel, Großcompiler-Sprachen im "multitasking"- bzw. "multiprogramming"-Betrieb anzuwenden, ist man in den Bereich von Systemgrößenordnungen eingestiegen, in denen das Speicherwerk von einer separaten Hardware organisiert und verwaltet wird. Das Z 8000-System offenbart hier seine Zielsetzung, indem es einen Einchip-Speicherorganisator mit einschließt. Das Spektrum der Systemgröße eines Z 8000-Computers beginnt von einem sprichwörtlichen Kleinsystem, wie es in Abschnitt 4.9 vorgestellt wurde, und endet bei extensiven Hostcomputersystemen und Parallelrechnern höchster Prägung mit Speicherfeldern im 50 Megabytebereich. Der Unterschied in der Zielsetzung eines Z 8000-Systems gegenüber den bekannten 8-bit-Systemen wie Z 80, 8080, 6800, 6502 . . . zeigt sich, abgesehen vom Datenformat, in drastischer Weise speziell darin, daß es für die genannten 8-bit-Systeme keine Einchip-Speicherorganisationseinheit wie die Z 8010 MMU gibt. Die 8001-CPU mit der Z 8010 MMU ergänzt sich zu einem Potential von Eigenschaften, die noch bis zu Anfang der 80er Jahre ein ausschließliches Privileg der Basisebene absoluter Großrechner war. Der einzige Unterschied zwischen Großrechnern und der Z-Kombination besteht im Datenformat. 8-bit-Mikrocomputer haben mit dem Potential dieser Eigenschaften wirklich nichts mehr zu tun. Hier trennen mehrere Größenordnungen.

Die Zielsetzung eines Z 8000-Systems mit Speicherverwaltung ist folgende: In einem Mehrfachzugriffsystem treten die Probleme bei einem Zugriff auf das Hauptspeicherwerk auf. Mehrere Anwender des Systems melden einen Anspruch auf dieses Hauptresource über ihre Terminals an. Der Dialogbetrieb mit dem Rechner erfolgt nach dem Zeitschnittverfahren und betrifft das Erstellen, Compilieren, Testen und Anwenden des Programms. Die Problematik liegt in dem nicht genau vorhersagbaren Speicherbereich, den ein einzelner Anwender benötigt. Ein Anwender bekommt normalerweise den Anspruch auf einen zahlenmäßig fest zugeteilten Bereich des Speicherfeldes. Der Nachteil dieser Organisation ist leicht verständlich. Entweder kommt der Anwender mit der gegebenen Kapazität aus, läßt womöglich noch Restkapazität übrig, oder er muß scheinbar "gewaltsam" versuchen, seine Programmieraufgabe so hinzubiegen, daß eine Überschreitung seines Speicherbereichs nicht erfolgt. Daß dies nicht immer einfach ist, läßt sich denken. Auf der Suche nach Abhilfe des Problems der starren Hauptspeicherzuweisung mußte man zwangsläufig auf eine neue, bessere Lösung stoßen. Man muß das zur Verfügung stehende gesamte Speicherfeld dynamisch in seinen Benutzergrenzen verschieben können, d.h. die logischen Hauptspeicheradressen müssen in ein physikalisches Feld umgesetzt werden, in dem es die starre Koordinierung logischer Adreßgrenzen nicht mehr gibt. Jeder Anwender soll auf ein homogenes, logisches Speicherfeld Zugriff haben, in dem die Bereichszuteilung variabel nach Bedarf des Anwenders von einer Speicherorganisations-Einheit organisiert wird.

Die Zuteilung erfolgt dynamisch und ist im Betriebssystem festgelegt. Die variable Speicherfeldzuteilung nach den Erfordernissen des Anwenders bedingt zur Software eine hinreichende Hardware. Eine gelungene Synthese aus Speicherorganisation und Betriebssoftware bewirkt eine sehr effektive Speicherausnutzung und läßt somit unbenutzte Restkapazitäten wie auch den Zwang auf bestimmte Programmumfänge nicht mehr auftreten. Der durch das Betriebssystem definierte virtuelle Adreßraum für einen Anwender ermöglicht die Überlagerung von Quelldateien, die von mehreren Anwendern in ihrem Programm benutzt werden, ohne daß für den Einzelnen eine Umrechnung in die effektive Hauptspeicheradresse erfolgen muß. Die Umrechnung erfolgt selbstverständlich, jedoch ohne daß der Anwender etwas dazu tun muß. Er benutzt gemeinsame, Datenressourcen mit seinen logischen Adressen. Das ist ein echter Vorteil. Man nennt dieses Prinzip die virtuelle Adressierung. Diese Möglichkeit eröffnet der Speicherorganisations-Baustein Z 8010 MMU mit einer relativ überschaubaren Betriebssoftware und einem sehr einfachen Hardwareanschluß an die CPU. Der Aufbau von "multi-user"-Rechnersystemen wird mit der MMU ziemlich einfach. Genau diese Eigenschaft war bisher ein Privileg relativ umfangreicher und teurer Systeme.

Die Zielsetzung des MMU-Bausteins ist nach dieser ersten Einführung etwas leichter verständlich. Die wesentlichen Eigenschaften lassen sich in folgende Einzelheiten gliedern:

- Umsetzung einer logischen CPU-Adresse in eine physikalische Adresse
- Homogene Adressierung in Form eines 24-bit-Adreßwertes
- Dynamische Verschiebung eines logischen CPU-Segments im gesamten physikalischen 16 MB-Feld
- Variable Größendefinition eines Anwendersegments von 256 Byte bis 64 KB in 256 Byte-Schritten
- Vielfältige Schutzdefinition eines Anwendersegments: nur lesen, nur im System-Modus zugriffsfähig, nur im Programm-Ausführungsmodus zugriffsfähig, Segmentschutz vor DMA-Zugriff, Segmentschutz vor CPU-Zugriff (DMA-Segment)
- 64 frei definierbare Segmente unter der Organisation einer MMU, wahlweise von den 128 CPU adressierten Segmenten die ersten 64 oder die letzten 64 Segmente
- Mehrfachanwendung von MMUs in einem Z 8000-System
- Genereller Schutz vor willkürlichem Überschreiten eines Datensegments
- Schutz vor unberechtigtem Zugriff in das Betriebssystem oder Teile davon
- Hardware-Synchronisation vor einem Zugriff des DMA-Steuerbausteins auf ein Segment

5.1 PRINZIPIELLE FUNKTION DER Z 8010 MMU

Die MMU ist kein Prozessor. Die MMU (Memory Management Unit) ist eine teils festverdrahtete und teils Random-Zugriffslogik, die auf den Busmechanismus des Z 8001 optimal zugeschnitten ist. Die MMU besteht aus einem Adreß-/Daten-Businterface, einem Segment-Beschreibungsregisterfeld, (64 RAM-Register mit 32-bit-Format), einem 16-bit-Adreßaddierwerk und einer Prüf- und Vergleichslogik, die im Subnanosekundenbereich feststellt, ob bei einer einge-

gebenen, logischen Adresse der CPU irgend eine Verletzung der Zugriffsbedingungen auf den Segment-Speicherplatz auftritt. In einem 8-bit-Eigenschaftsfeld eines Segment-Beschreibungsregisters werden die Zugriffsbedingungen festgelegt. Der angestrebte Speicherzugriff der CPU wird von der MMU genau überprüft (Adresse, schreiben, lesen), ob die CPU dies auch darf. Ein Speicher-Segment wird nicht nur durch seine Anfangsadresse und die Ausdehnung im physikalischen Adreßfeld beschrieben, sondern noch durch eine Reihe von Bedingungen in Form von Flags, die insgesamt die Definition eines Segments ergeben. Über die Einhaltung der Bedingungen achtet peinlich genau die Prüf- und Vergleichslogik. Die Bedingungen für einen erlaubten Zugriff in das Segment muß man sich vor der MMU-Initialisierung selbst definieren. Das hat für alle 64 Segmente zu geschehen. Die interne Logik der MMU wird durch eine Reihe von Status- und Steuerregister so transparent gehalten, daß die exakte Steuerbarkeit und Kontrolle von der CPU her gegeben ist. Eine Reihe, bereits genannter, spezieller E/A-Befehle der CPU ermöglichen einen schnellen Datenaustausch bezüglich Segmentbeschreibungen.

Wollte man sich eine MMU diskret nachbauen, könnte man das wohl anstellen. Nach dem Bustrenn-Interface könnte mit dem Adreßaddierwerk begonnen werden. Hat man das obere Byte der CPU-Offsetadresse stellenrichtig anliegen, muß als zweiter Operand die Segmentanfangsadresse am Addierer anliegen. Woher nehmen, wenn nicht stehlen ? Stehlen geht auch nicht. Man muß die Segmentanfangsadresse aus der logischen Segmentadresse SN0 . . . SN6 der CPU konstruieren. Übernimmt man sie direkt, wäre eine Addition unsinnig. Wünscht man eine stetige Verteilung der Segmente im physikalischen Adreßraum (Ende Segment n, nächste Adresse: Anfang Segment n + 1), kann man sich die Addition in der Tat sparen. Der Kernpunkt der Umsetzung logischer CPU-Segmente ist, eine variable Verteilung variabler Segmentgrößen im physikalischen Adreßraum zu erreichen (einschließlich der Überlappung zweier oder mehrerer Segmente). Die Zuordnung einer Segmentanfangsadresse im physikalischen Adreßraum bezüglich einer ausgegebenen logischen Segmentadresse der CPU ist entscheidend. Damit wird man beim Selbstbau schon vor einige Probleme gestellt. Vollends problematisch wird es, wenn die Bedingungen eines definierten Segments rekonstruiert und gescheckt werden sollen. Spätestens ab hier ist es besser, man gibt den Selbstbau auf, macht es sich leichter und verwendet die MMU.

In **Bild 5.1** ist vereinfacht der interne Aufbau der Z 8010 MMU dargestellt. Über das A/D-Trenninterface gelangt das obere Adreßbyte der Offsetadresse an das Addierwerk und zugleich in die Vergleichslogik. Die 7-bit-Segmentadresse der CPU adressiert direkt das Feld der 64 Segmentbeschreibungsregister, wobei genaugenommen auf den Basisadreßteil der Beschreibungsregister abgezielt wird. Hier findet die grundsätzliche Übersetzung oder auch Zuordnung statt, wie sie vorhergehend genannt wurde. Eine bestimmte Segmentadresse der CPU wirkt in der MMU als Adresse für das Basisfeld, liest den 16-bit-Übersetzungswert aus und legt ihn an den Eingang des Addierwerks an. Mit der CPU-Segmentadresse werden zugleich das Größen- und Eigenschaftsfeld (je 8 bit) ausgelesen und von der Prüf- und Vergleichslogik übernommen. Der Vergleich von Status der CPU, Größenfeld, Flags des Eigenschaftsfeldes und zuletzt (besser zugleich) des Offsetadreßwerts (A8 . . . A15) der CPU ergibt als Ausgabe einen Segment-Trap, wenn von der CPU ein unzulässiger Zugriff in den Speicher erfolgt. Ist der CPU-Zugriff korrekt, bleibt $\overline{\text{SEGT}}$ gleich Hi. Mit diesem MMU-Interrupt-Signal erfolgt zugleich die Aktivierung des Suppress-Signals an das Speicherwerk (Schreibsperre). Die Ausgabe der physikalischen Adresse erfolgt etwa nach 140 ns nach Beginn des ersten Maschinentakts eines CPU-Transferzyklus. Die Adreßleitung

SN6 wird nach einer Abfrage mit dem URS-Flag im Steuermode-Register verglichen und festgestellt, ob die Segmentadresse auf die Segmente 1 . . . 64 oder 65 . . . 128 abzielt. Der Wert von SN6 gilt damit auch als gewisser Chipselekt einer MMU, je nachdem, welche Segmente zahlenmäßig angewählt sind.

Bild 5.2 zeigt genauer das Detail der Umsetzung einer Segment- und Offsetadresse der CPU zu einem physikalischen Adreßwert. Zeigt die Segmentadresse den Wert 1010100 = 54H (= Segment 88)

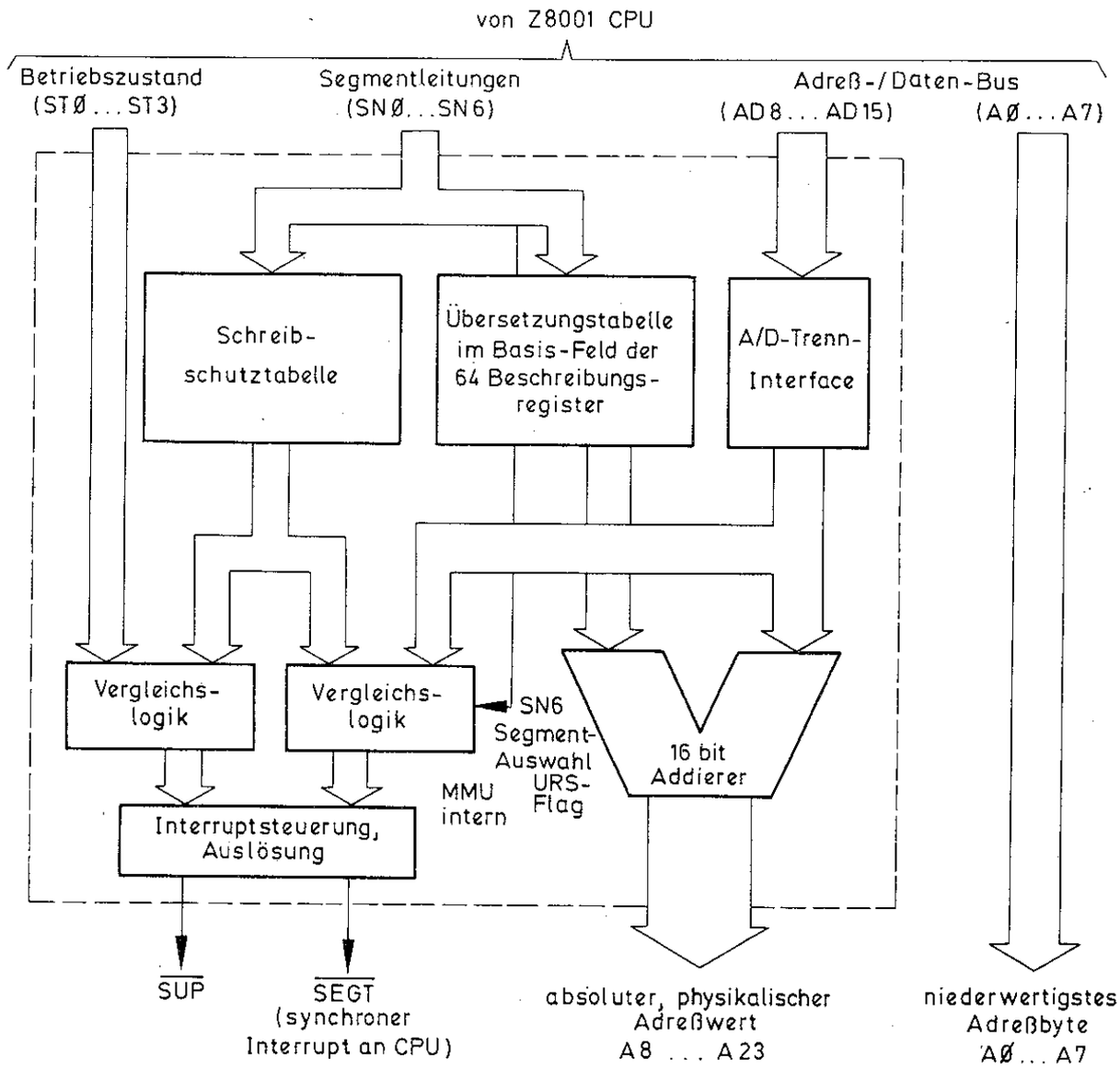


Bild 5.1

an, wird dieser Wert um 40H verringert, so daß die eigentliche Adresse für das Beschreibungsregister 54H – 40H = 14H lautet. SN6 ist, wie gesagt, gleich 1. Damit muß das URS-Flag gleich 1 sein, sonst fühlt sich die MMU nicht aktiviert (A8 . . . A23 hochohmig). Ist das URS-Flag gleich 1, wird grundsätzlich von jeder ankommenden Segmentadresse der CPU 40H (= 64) abgezogen. Der verbleibende Rest (hier 14H) ist dann folglich die Adresse für das Beschreibungsregister. Ist SN0 . . . SN6 gleich oder kleiner als 40H und das URS-Flag gleich Null,

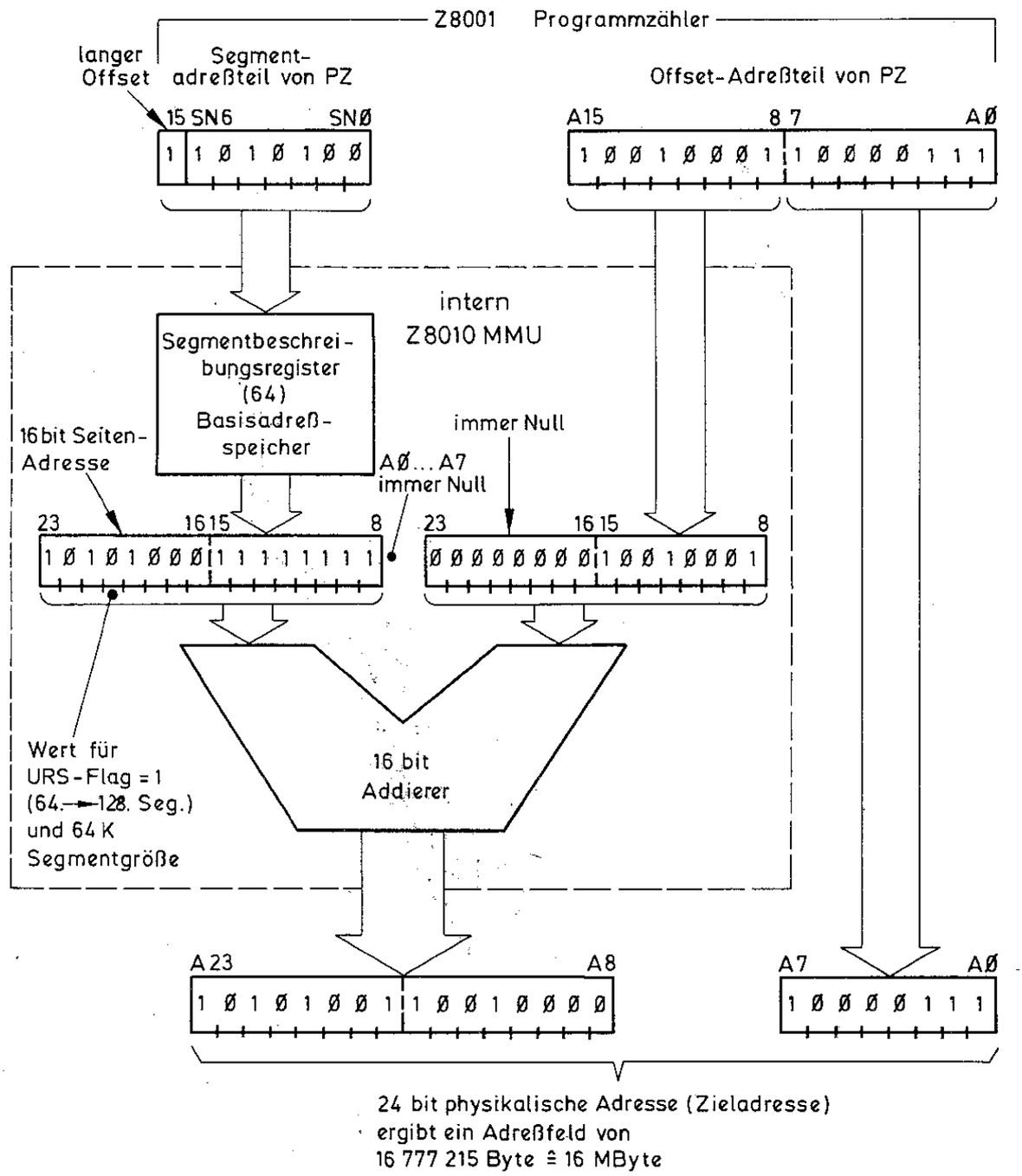


Bild 5.2

wird die Konstante nicht abgezogen. In diesem Fall ist die CPU-Segmentadresse unmittelbar die Adresse für das Beschreibungsregister. Dieser Vorgang läuft in der MMU in ein paar Nanosekunden ab. Hiervon merkt ein Anwender überhaupt nichts. Nach der Adressierung des Basisfeldes wird der Inhalt an 14H ausgelesen und an die Eingänge des internen Addierwerks gelegt. Nehmen wir an, aus dem Basisfeld wird 1010|1000|1111|1111 = A8FFH ausgelesen (Initialisierung des Programmierers), so ist diese Adresse die Anfangsadresse eines Segments im physikalischen Adreßraum. Die Offsetadresse 0 . . . 0|1001|0001 = 0091H gibt die Distanz an. Da nun dieser Wert um 8 Stellen (rein rechnerisch) nach links verschoben ist, ergibt erst die Summe mit dem Adreßbyte A0 . . . A7 den endgültigen Zielwert, der mit A99087H als physikalische Zieladresse in Segment 88 ausgegeben wird. Wie schon gesagt, das Basisfeld ist Aufgabe des Programmierers. Er muß bestimmen, welche Anfangsadresse Segment 88 bekommen soll.

Dies ist der grundsätzliche Vorgang einer Adreßumsetzung bzw. Berechnung. Daß dies Zeit benötigt, wurde bereits angedeutet. Der Zeitbedarf ist jedoch trotzdem so gering, daß sich die erforderliche Zugriffszeit eines Speichers, der über die MMU adressiert wird, auf 200 ns verringert, sollte kein Wartezyklus beim Zugriff eingeschoben werden.

Es gibt zwei Formen der Segmentadressierung, die sich mit der Bezeichnung Kurzsegment-Adressierung und Langsegment-Adressierung unterscheiden. In der Kurzsegment-Adressierung ist ein Segment nicht länger als 256 Byte. Ein langes Segment ist üblich mit einem 16-bit-Offset versehen und ergibt maximal 64 KB. Der Unterschied wird rein softwaremäßig getroffen, indem beispielsweise in der direkten Adressierungsart Segment- und Offsetadresse in einem Wort steht. Das höchstwertige Bit (MSB) des Adreßwortes wird immer Null gesetzt, wenn Kurzsegmente adressiert werden.

Bild 5.3 zeigt den Unterschied. Für die MMU wirkt sich das in der Form aus, daß der Offsetadreßoperand am Eingang des Adreßaddierwerks von Stelle 8 . . . 23 lauter Nullen enthält.

Um sich unter dem Beschreibungsregister eine bessere Vorstellung machen zu können, sind in **Bild 5.4** die einzelnen Elemente dieses 32 bit langen Registers gezeigt. In dem besagten Basis-Feld von 16 bit Länge steht die Anfangsadresse des Segments im physikalischen Speicherraum. Dieser Wert ist frei wählbar. Im praktischen Fall wird er in einen sinnvollen Bezug zur Lage anderer Segmente gebracht. Das ist problem-bedingt und ist vom Programmierer zu definieren. Die Segmentgröße wird in dem 8 bit langen Limit-Feld definiert. Das Intervall der Größenänderung ist mit 256 Byte gegeben. Steht in dem Registerteil der Wert 25H, bedeutet dies 38 Bünde zu je 256 Byte. Das Segment hat somit 9728 Byte. Die Offsetadresse für dieses Segment erlaubt nur Werte zwischen 0H . . . 25FFH. Ein Wert für N ergibt immer N + 1 Bünde zu 256 Byte. Ist die Offsetadresse mit 2560H ausgegeben, kann ein Trap auftreten. Es hängt davon ab, wie die nachfolgende Adresse von 25FFH geschützt ist (definiert in den Flags). Eine Schreibwarnung tritt bereits zwischen 2500H . . . 25FFH auf, wenn das Segment als Stapelsegment gekennzeichnet wurde. In diesem Fall ist das DIRW-Flag gesetzt worden und die Schreibwarnung erzeugt in jedem Fall einen Trap ($\overline{\text{SEGT}}$ gleich Lo). Ist das SLV-Flag in einem der Statusregister gesetzt worden, wird bei der Adresse 2560H sicher ein Trap erzeugt. In dem 8 bit langen Eigenschaftsfeld schließlich wird die Definition für ein einziges Segment in 8 verschiedenen Flags definiert. Allein 5 dieser Flags dienen ausschließlich als Schutzmaßnahme gegen einen unberechtigten oder willkürlichen Schreib- oder Lesezugriff. Es sei hier nur nebenbei bemerkt, daß diese Eigenschaft der MMU ersatzweise durch Software wie auch Hardware zu simulieren als sehr schwierig anzusehen ist.

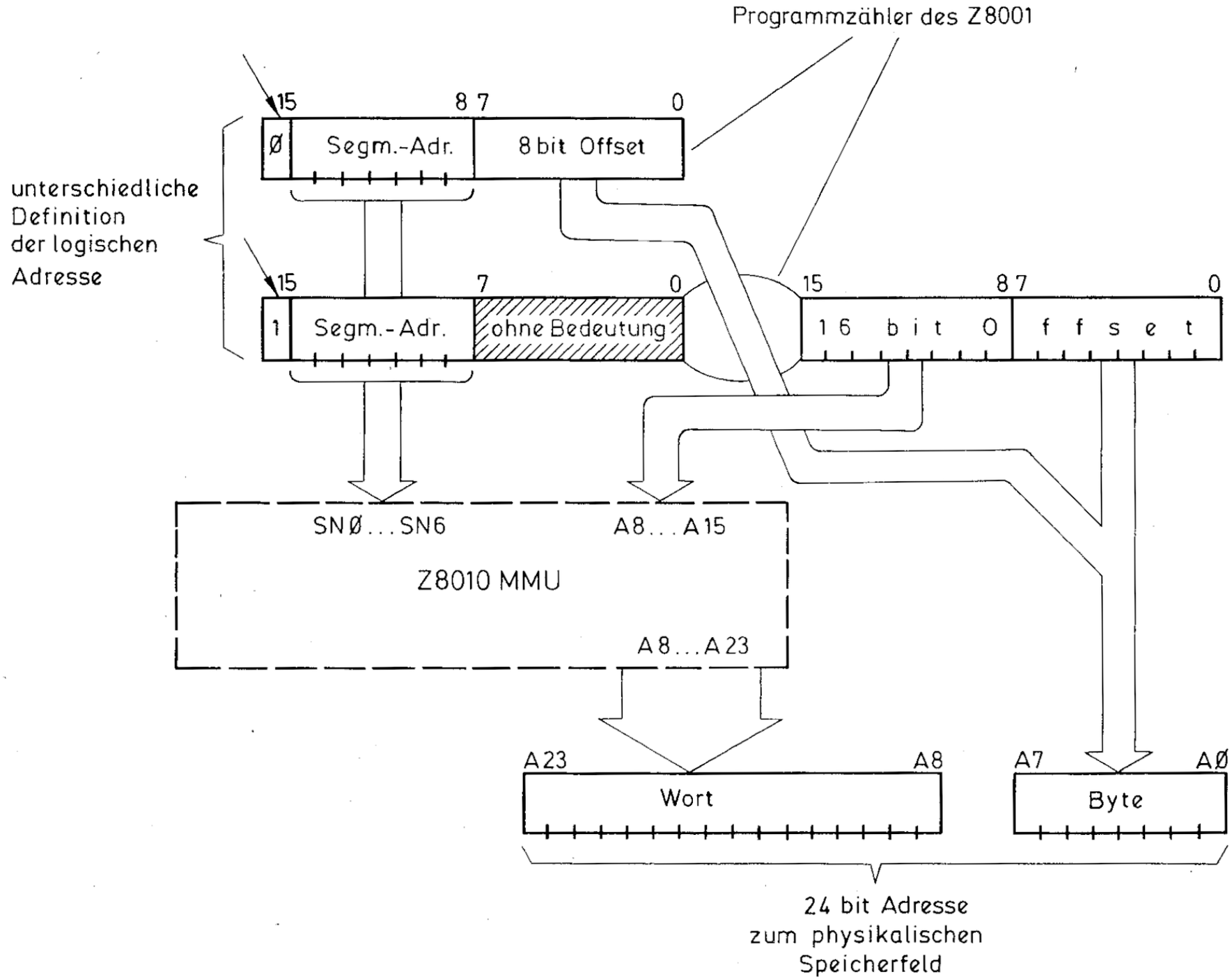

Programmzähler des Z8001
15
8 7
0
Ø
Segm.-Adr.
8 bit Offset
unterschiedliche
Definition
der logischen
Adresse
15
7
0
1
Segm.-Adr.
ohne Bedeutung
15
8 7
0
1 6 b i t O
f f s e t
SNØ...SN6
A8...A15
Z8010 MMU
A8...A23
A23
A8
Wort
A7
AØ
Byte
24 bit Adresse
zum physikalischen
Speicherfeld

Bild 5.3

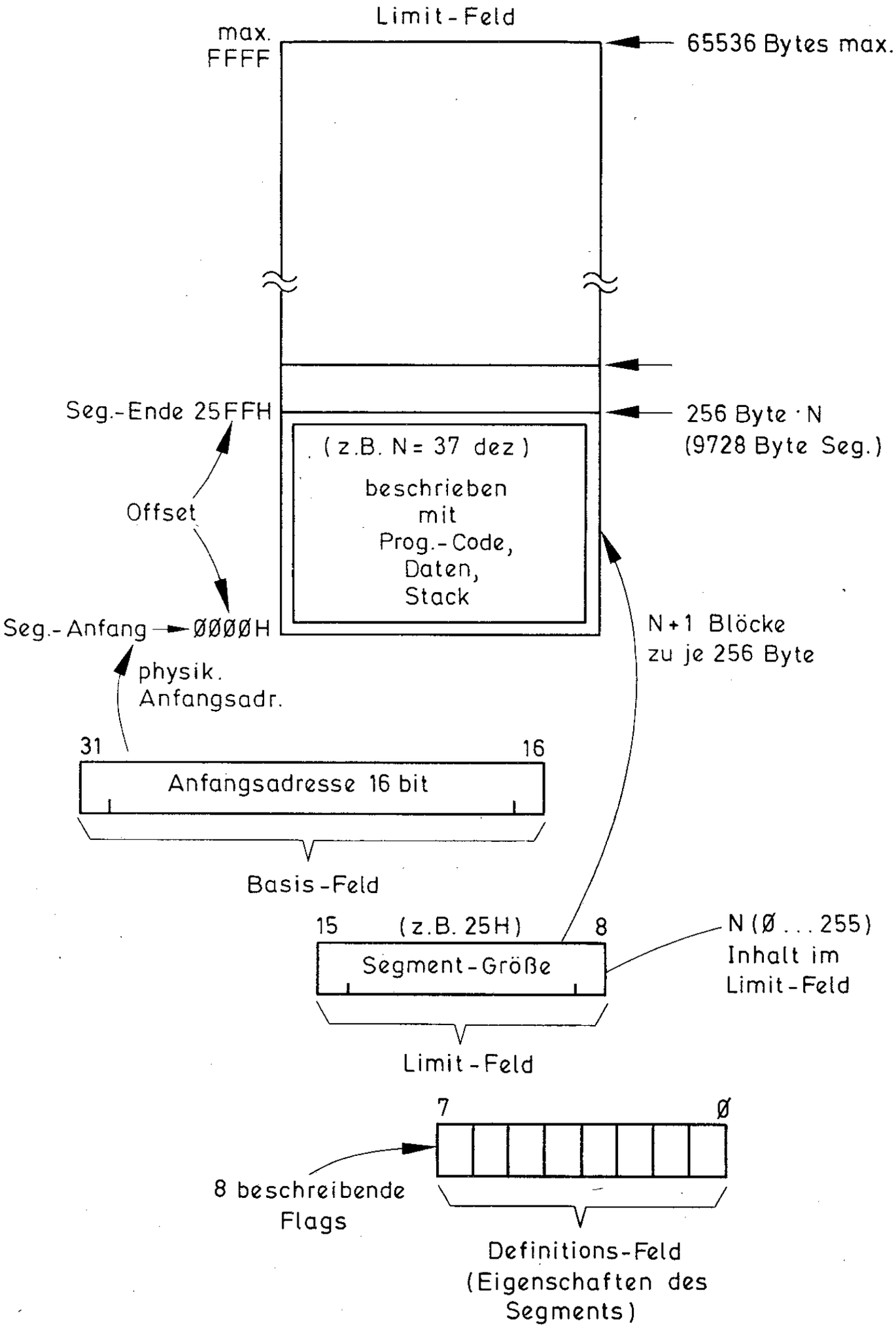

Limit-Feld
max.
FFFF
65536 Bytes max.
Seg.-Ende 25FFH
256 Byte · N
(9728 Byte Seg.)
(z.B. N = 37 dez)
beschrieben
mit
Prog.-Code,
Daten,
Stack
Offset
Seg.-Anfang
ØØØØH
N+1 Blöcke
zu je 256 Byte
physik.
Anfangsadr.
31
16
Anfangsadresse 16 bit
Basis-Feld
15
(z.B. 25H)
8
Segment-Größe
N (Ø ... 255)
Inhalt im
Limit-Feld
Limit-Feld
7
Ø
8 beschreibende
Flags
Definitions-Feld
(Eigenschaften des
Segments)

Bild 5.4

5.1.1 MMU-Register und Initialisierung

Das gesamte Registerwerk des Z 8010 wird in das Feld der 64 Segmentbeschreibungsregister (256 Byte), der 3 Steuerregister (3 Byte) und der 6 Statusregister (6 Byte) unterteilt. Die Beschreibungsregister können im Lese-/Schreibmodus behandelt werden, ebenso die drei Steuerregister. Die Statusregister sind nur lesbar, wobei das Register für die Anzeige der Verletzungsbedingungen (Violation Type Register) eines unzulässigen Speicherzugriffs der CPU auf Null gesetzt werden kann.

Da die Aktivierung einer MMU besonders von den gesetzten Flags im ersten der drei Steuerregister, dem Betriebsart- oder Moderegister abhängt und ferner die beiden anderen unmittelbar auf die Beschreibungsregister bezogen sind, muß man zwischen Segmentbeschreibung und MMU-Steuerung einen engen Zusammenhang sehen. In **Bild 5.5** sind beide Registerarten zusammen dargestellt. In der folgenden Aufstellung werden die Flags des Definitionsfeldes im Segmentbeschreibungsregister vorgestellt. Die Reihenfolge ist in der steigenden Stellennummer angegeben.

RD-Flag: Ist dieses Flag gesetzt, bedingt eine CPU-Operation in dem Segment nur die Zulassung eines Lesezugriffs. Schreiben ist unzulässig. Die MMU antwortet nach Abfrage des R/$\overline{W}$-Eingangs bei einem Schreibzyklus durch die Ausgabe eines Suppress-Signals. Ist diese Leitung am Speicherwerk verdrahtet, kann damit das Schreibsignal hardwaremäßig unterdrückt werden. Man kann damit eine wichtige Datei oder ein Programm nicht überschreiben, sei es willkürlich oder aus Unachtsamkeit.

SYS-Flag: Dieses Flag kann sehr bedeutsam sein. Ist dieses Flag gesetzt, darf ein Zugriff der CPU in dieses Segment nur im System-Modus erfolgen (N/$\overline{S}$ an der MMU wird abgefragt), d.h. dieses Segment ist privilegiert. Da im System-Modus generell alle Initialisierungen des Systems möglich sind, kann man mit dieser Vorgabe sehr gezielte Korrekturen in kritischen Bereichen der Betriebssoftware unterbinden. Ein Zugriff im Normal-Modus in solch ein Segment erzeugt einen Trap, das die CPU (nach entsprechender Softwarevorbereitung der Serviceroutine) zu der Ausgabe am Terminal veranlassen könnte: "Freund, du darfst dieses Segment nur im System-Mode benützen". Handelt es sich beispielsweise um eine Ausgaberoutine, Speicherinhalt auf Drucker ausgeben, kann diese Routine nur angewendet werden, wenn über das Terminal ein System-Mode-Zugriff möglich ist. Im Gegensatz zu einzelnen CPU-Befehlen wird hier ein komplettes Programm privilegiert.

CPUI-Flag: Dieses Flag bewirkt im gesetzten Zustand, daß die CPU in dieses Segment generell keinen Zugriff hat. Dieses Segment kann nur von einem DMA-Baustein belegt werden, wie den Z 8016 DMATC. Angezeigt wird eine Adresse eines DMA-Bausteins durch das Lo setzen des DMA-SYNC-Eingangs der MMU, das vom DMA-Baustein ausgelöst wird (speziell Z 8016). Man könnte ein DMA-Segment als Pufferdatenstapel ansehen, der für eine Fernübertragung vorbereitet ist. Die Daten sind bereits behandelt und aus mehreren verschiedenen Datensegmenten in dem DMA-Segment zusammengestellt worden. Dieses Flag setzt einen Trap, wenn die CPU einen Zugriff in das Segment anzeigt.

EXC-Flag: Ist dieses Flag gesetzt, bedeutet das einfach, daß dieses Segment nur im Zuge einer Programmausführung benutzt werden darf. Die MMU fragt zu diesem Zweck die ST0 . . . ST3-Eingänge ab und kontrolliert, ob die Statuswerte Nr. 12 und 13 bei

einem Lesezugriff auftreten. Ist das nicht der Fall, erzeugt die MMU einen Trap. Programme in dem Segment können aus diesem Grund nicht ausgelesen werden, um z.B. die Codes zu studieren. Man kann die Programme vor einer Kopie schützen. Gibt man bei einer Diskübertragung dieses Segments ein File mit "nur ausführen" vor, wird ein entsprechendes Betriebssystem mit der Übertragung zugleich eine Initialisierung des betreffenden Beschreibungsregisters (EXC-Flag gleich 1) verbinden.

DMAI-Flag: Ist dieses Flag gesetzt, wird der MMU-Eingang DMASYNC auf Lo abgefragt. Ist dies der Fall, weiß die MMU, daß der Zugriff von einem DMA-Baustein erfolgt und zeigt durch die Ausgabe eines Suppress-Signals eine Unterdrückung des Zugriffs an. Der DMA-Baustein Z 8016 hat einen Eingang für das $\overline{SUP}$-Signal vorgesehen.

DIRW-Flag: Ist dieses Flag gesetzt, wird trotz eines zulässigen Zugriffs der CPU ein Trap erzeugt, wenn sie eine Adresse ausgibt, die gerade in den letzten 256 Byte-Bund vor Segmentende hineinfällt. Dieser Trap zeigt eine Schreibwarnung an und soll andeuten, daß das Ende des Speicherplatzes für das Segment "in Sicht" ist. Dies ist besonders bei Stapelsegmenten wichtig, weil hier relativ leicht ein Überlauf auftreten kann. Wird vor einer Anzahl Programmsprünge eine größere Zahl von aktuellen Programmzuständen in den Stapel geschoben (übliche Rettvorgänge), ist ein Überlauf eines solchen Segments denkbar. Der Trap soll zu einer Anzeige am Terminal einleiten, d.h. "vorsicht Stapelspeicherende". Ein Suppress wird von der MMU bei diesem Flag nicht ausgegeben.

CHG-Flag: Dieses Flag wird automatisch gesetzt, wenn die CPU oder ein DMA-Baustein in das Segment schreibt (Abfrage des R/$\overline{W}$-Eingangs auf Lo), überschreibt oder dazuschreibt. Man kann somit nach jedem Schreibzyklus feststellen, ob in einem Segment eine Manipulation vorgenommen wurde. Dies ist dann nützlich, wenn dieses Segment z.B. von einem DMA oder einer externen CPU beschrieben wird. Eine Manipulation eines Segments nimmt man in der Regel dann vor, wenn man das Original auf einer Diskette zur Verfügung hat. Es könnten sich z.B. variable Daten in dem Segment befinden, die von Zeit zu Zeit auf die Diskette geschrieben werden. Die Anzahl der Manipulationen wird durch Zählen des aktiven CHG-Flags meßbar.

REF-Flag: Dieses Flag wird automatisch gesetzt, wenn ein Zugriff auf ein Segment erfolgt ist. Dabei ist es gleichgültig, ob der Zugriff von der CPU oder einem DMA-Baustein im Lese- oder Schreibzyklus erfolgt ist. Bedingung dabei ist, daß kein unzulässiger Zugriff erfolgt, d.h. es darf kein bisher beschriebenes Flag (wenn es gesetzt ist) bezüglich seiner Bedingung verletzt werden. Mit der Abfrage dieses Flags ist relativ einfach feststellbar, wie oft auf das betreffende Segment nach einer bestimmten Betriebszeit zugegriffen wurde.

Die ersten 5 Flags spezifizieren sehr genaue Bedingungen bzw. Eigenschaften des Segments. Setzt man alle 5, könnte man das fast als Unfug bezeichnen, denn bei einem in der Art spezifizierten Segment wird man praktisch keine Möglichkeit finden, berechtigt zuzugreifen. Hier muß ein Trap erscheinen. Die Definition will also überlegt sein. Die automatisch aktivierten Flags REF und CHG dienen auch als Parameter für eine Statistik. Es sei nochmals darauf hingewiesen, daß das Eigenschaftsfeld für jedes Beschreibungsregister zu definieren ist, also 64 mal 6 Flags.

Es können mehrere MMUs in einem System integriert werden. Die drei Steuer-

register definieren sehr präzise, wann und wie eine MMU im System aktiviert wird. Dies betrifft den regulären Zugriff auf ein Segment über die MMU, womit das Betriebsart-Register angesprochen ist. Die beiden anderen Register treten bei der Initialisierungsphase in Aktion. Die folgenden Flags des Betriebsart-Registers werden wieder in steigender Stellenzahl vorgestellt.

Identifikation: Dieses 3-bit-Feld bezeichnet logisch eine MMU von maximal 8 möglichen, die in ein Z 8001-System integriert werden können. Die Auswahl einer MMU erfolgt für eine Initialisierung an dem $\overline{CS}$-Eingang. Hat man ID2 . . . ID0 auf 001 gesetzt, legt man an $\overline{CS}$ die CPU-Busleitung AD1 an. Ist AD1 gleich Lo, ist diese MMU aktiv. Wird die Identifikation auf 011 gesetzt, nimmt man AD1 und AD2, verknüpft sie in einem UND-Gatter und legt den Ausgang an $\overline{CS}$ an. Macht man das so, muß der Ausgang invertiert an $\overline{CS}$ der zweiten MMU angelegt sein. Im Prinzip geht das wie bei der Speicherdekodierung; für CS darf es nur eine Adresse geben. Am einfachsten ist es, speziell bei mehreren MMUs im System, man verknüpft AD1 ... . AD3 mit den Eingängen A, B und C eines LS138, legt an G1 AD0, an $\overline{G2A}$ den Status Nr. 3 und $\overline{G2B}$ statisch auf Lo. Die Initialisierungsanwahl ist damit gesichert, wenn an jeden Y-Ausgang eine $\overline{CS}$-Leitung einer bestimmten MMU angebunden ist. In einem Segment-Trap-Anerkennungszyklus ist eine auf AD8 . . . AD15 ausgesandte Kennung zugleich auf die MMU zu beziehen, die den Segment-Trap ausgelöst hat. Wenn die MMU neu definiert werden soll, muß die CPU die AD-Kombination ausgeben, die auf Grund der Kennung die entsprechende MMU freigibt.

NMS-Flag: Ist dieses Flag gesetzt, bezieht sich das mit dem MST-Flag auf eine Mehrfach-Konfiguration im System. Ist MST gesetzt, entscheidet das NMS-Flag, ob die Übersetzung des Segments im Normal- oder Systemmodus erfolgen soll. Ist es auf 1 gesetzt, wird der MMU-Eingang N/$\overline{S}$ auf 1 abgefragt und die Übersetzung erfolgt im Normalmodus.

MST-Flag: Ist dieses Flag gesetzt, zeigt das an, daß mehrere MMUs im System integriert sind und demzufolge mehrere Übersetzungstabellen existieren. Das NMS-Flag trifft die Entscheidung, ob es eine Übersetzungstabelle für einen Normal- oder Systemmodus-Zugriff ist.

URS-Flag: Dieses Flag ist sehr wichtig. Ist es auf 1 gesetzt, liefert der Vergleich mit SN6 (Segmentadreßleitung der CPU), daß die MMU die Segmente 64 . . . 128 behandelt. Ist SN6 gleich Null, wird die MMU nicht aktiviert und ihre Adreßausgänge sind hochohmig. Ist URS gleich Null und SN6 ebenso, ist der Segmentbereich 1 . . . 64 aktiviert.

TRNS-Flag: Ist dieses Flag nicht gesetzt, wird die Übersetzung von der logischen zur physikalischen Adresse nicht ausgeführt (A8 . . . A22 → → AD8 . . . A15, SN0 . . . 6; A23 = Null). Die Adresse verläßt die MMU unverändert. Ist dieses Flag gesetzt, wird die eigentliche Übersetzung nach dem Wert im Beschreibungsregister von der MMU ausgeführt.

MSEN-Flag: Dieses Flag ergibt im gesetzten Zustand generell die Freigabe der gesamten Logikfunktion der MMU. Ist MSEN gleich Null, werden die Adreßausgänge in den hochohmigen Zustand versetzt, in dem sie verharren.

Die beiden verbleibenden Steuerregister sind in ihrer Funktion sehr einfach. Das Segmentadreßregister gibt an, welches der 64 Beschreibungsregister gelesen

oder beschrieben werden soll. Der DSC-Teil gibt an, welches Byte des Beschreibungsregisters angesprochen werden soll.

00 → höherwertiges Byte des Basis-Feldes
01 → niederwertiges Byte des Basis-Feldes
10 → Limit-Feld
11 → Definitions-Feld (Eigenschaften durch Flags)

Wie erfolgt nun die Initialisierung des Beschreibungsregisters? Benützt wird dazu das 6-bit-Segmentadreßregister, das als Zeiger zu einem der Beschreibungsregister dient. Der DSC-Teil ist ein Unterzeiger auf das jeweilige Byte. **Bild 5.6** zeigt den Typus einer automatischen Initialisierung eines Segment-Beschreibungsregisters. Für diesen Fall muß der privilegierte CPU-Befehl SOTIR (spezielles Laden der MMU mit Inkrementieren im wiederholenden Modus) verwendet werden. Zwei Formen der Programmierung der Byte-Reihenfolge einer Initialisierung ist möglich (DSC zeigt zyklisch auf): beginnend mit oberem Byte (Basis) und unterem Byte, in diesem Fall nur Basisfeld, oder oberes und unteres Byte (Basis), Limit und zuletzt Eigenschaftsfeld, in diesem Fall das komplette Segment-Beschreibungsregister. Erfolgt die Adressierung nicht automatisch inkrementieren, läßt sich der DSC, wie in Bild 5.6 dargestellt, beispielsweise auf das Eigenschaftsfeld setzen (11). Das Laden erfolgt dann mit dem SOUT-Befehl.

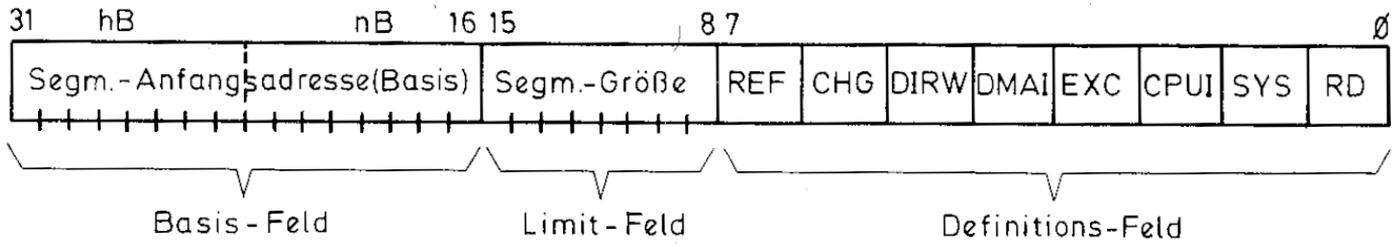
In **Bild 5.7** ist in einfacher Weise dargestellt, wie die Reihenfolge des Lesens bzw. Ladens eines MMU-Registers funktioniert. Man bestimmt zuerst den Befehl (1), entweder SIN oder SOUT (beide privilegiert), baut sich das Befehlswort zusammen (2) und setzt im Datenwort (3) auf das obere Byte AD8 . . . AD15 den Binärwert ein, der den Ansprüchen genügt. Man muß unbedingt darauf achten, daß die E/A-Adresse zur MMU gerade ist, weil das obere Byte geladen wird. AD0 wird für eine Freigabe einer MMU nicht benützt. Eine Zieladresse zur Freigabe ist **immer gerade.** Als niedrigstes Adreßbit nimmt man AD1. Verwendet man AD1 direkt, muß diese Leitung für $\overline{CS}$ Lo sein. Bei Verwendung des schon genannten LS138 als Dekodierer müssen die entsprechenden Leitungen, je nach Zieladreßwert, selbstverständlich Hi sein.

Nimmt man als Beispiel das Eigenschaftsfeld des Beschreibungsregisters 18 (Segment 18), das mit dem Definitionswert 02H geladen werden soll. Das folgende kleine Programm zeigt, wie das geht.

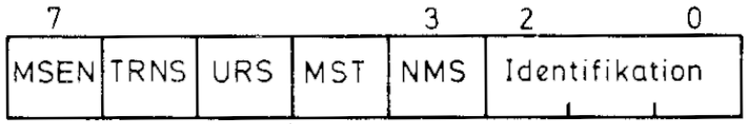
```
LDB     RH0 , #12       (laden 18)
LDB     RH1 , #03       (laden 3 für DSC)
LDB     RH2 , #02       (laden Definitionswert)
SOUTB   % 0100, RH0     (in MMU Segmentadreßregister)
SOUTB   % 2000, RH1     (in MMU DSC-Wert)
SOUTB   % 0A00, RH2     (in MMU Eigenschaftsfeld 18)
```

Die Reihenfolge muß man einhalten: laden Segmentadreßregister mit 18, DSC mit 3 und schließlich das Eigenschaftsfeld von 18 mit dem Definitionswert 02H.

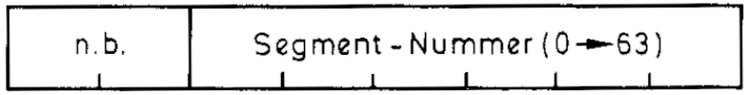
Zuletzt werden die sechs nur lesbaren Statusregister der MMU vorgestellt **(Bild 5.8).** Das Segmentdefinitionsregister als wichtigstes von allen gibt in Form von 8 Flags an, aus welchen Gründen bei einem laufenden Zugriff ein Segment-Trap entstanden ist. Ist irgendeine Verletzung der Bedingung bei einem Segment-Zugriff aufgetreten, wird das betreffende Flag gesetzt und kann als Referenz in einer Segment-Trap-Anerkennungsroutine benützt werden, um z.B. auf dem Bild-Monitor eine Meldung auszugeben. Das unmittelbar darunter liegende Register, Segmentadresse und Offset (nur oberes Byte), lokalisiert den Ort, an dem der Trap verursacht wurde. Es sei angemerkt, daß eine komplette Dokumentation der Trap-Ursache nur bei einem CPU-Zugriff in den Statusregistern abgelegt wird. Verursacht ein DMA-Baustein einen Trap, bleibt das Statusregisterfeld unverändert. Ein DMA-Trap ist, ausgenommen in dem Suppress-Signal, ohne Kennzeichnung. Das Buszyklus-Register speichert einen unzulässigen Zugriff mit Trap-Folge,



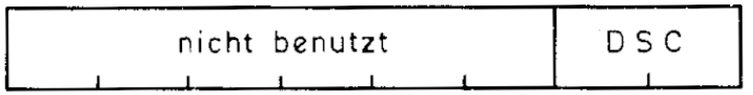
MMU - Segment - Beschreibungsregister



Betriebsart



Segment Adresse



Zähler für Zugriff der vier Bytes des Beschreibungsregisters

MMU Steuerregister

(Betriebsart wird über CPU-Busabschnitt AD8...AD15 gesetzt).

Bild 5.5

Zugriff auf Segment-Beschreibungsregister

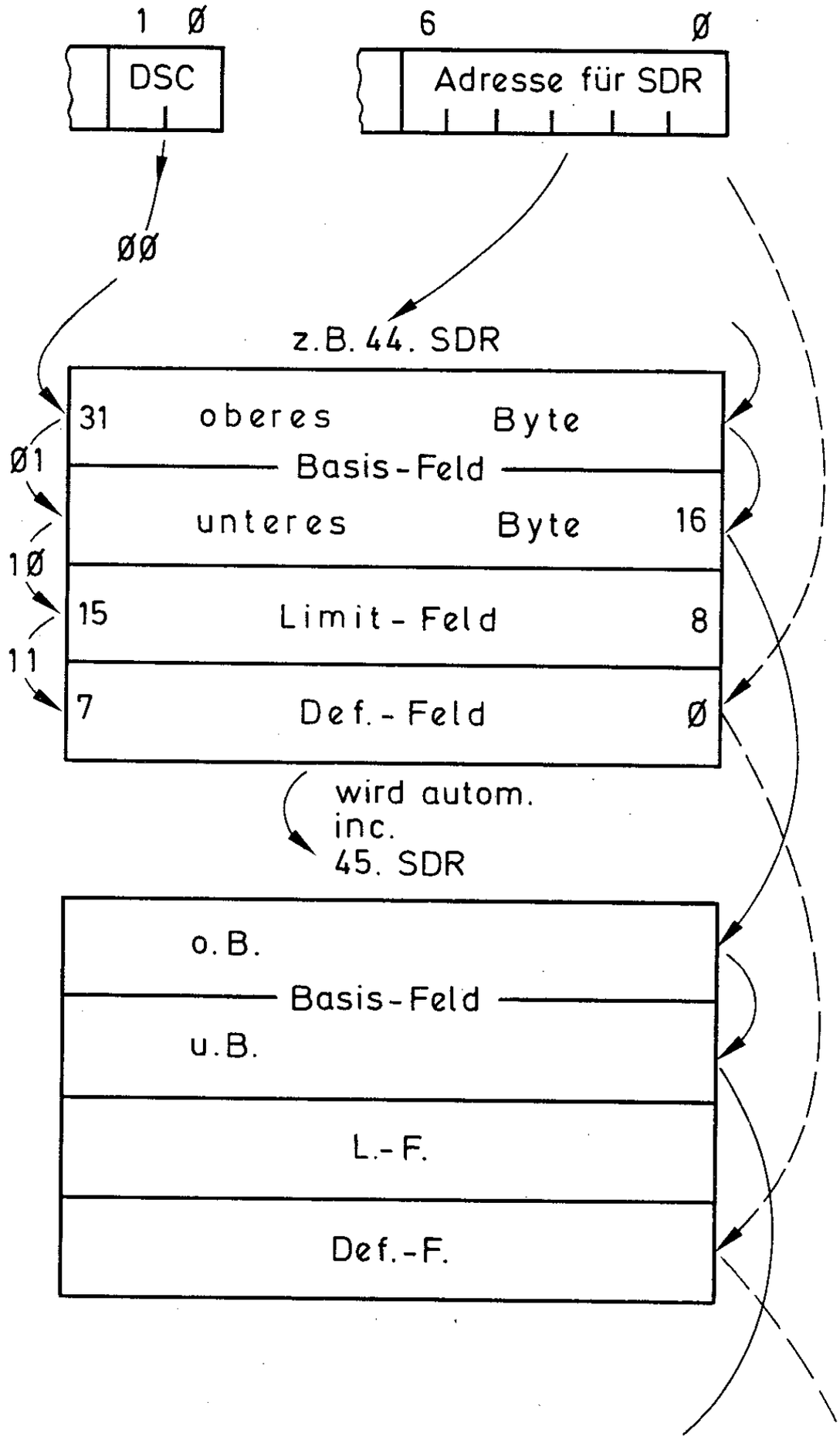


Bild 5.6

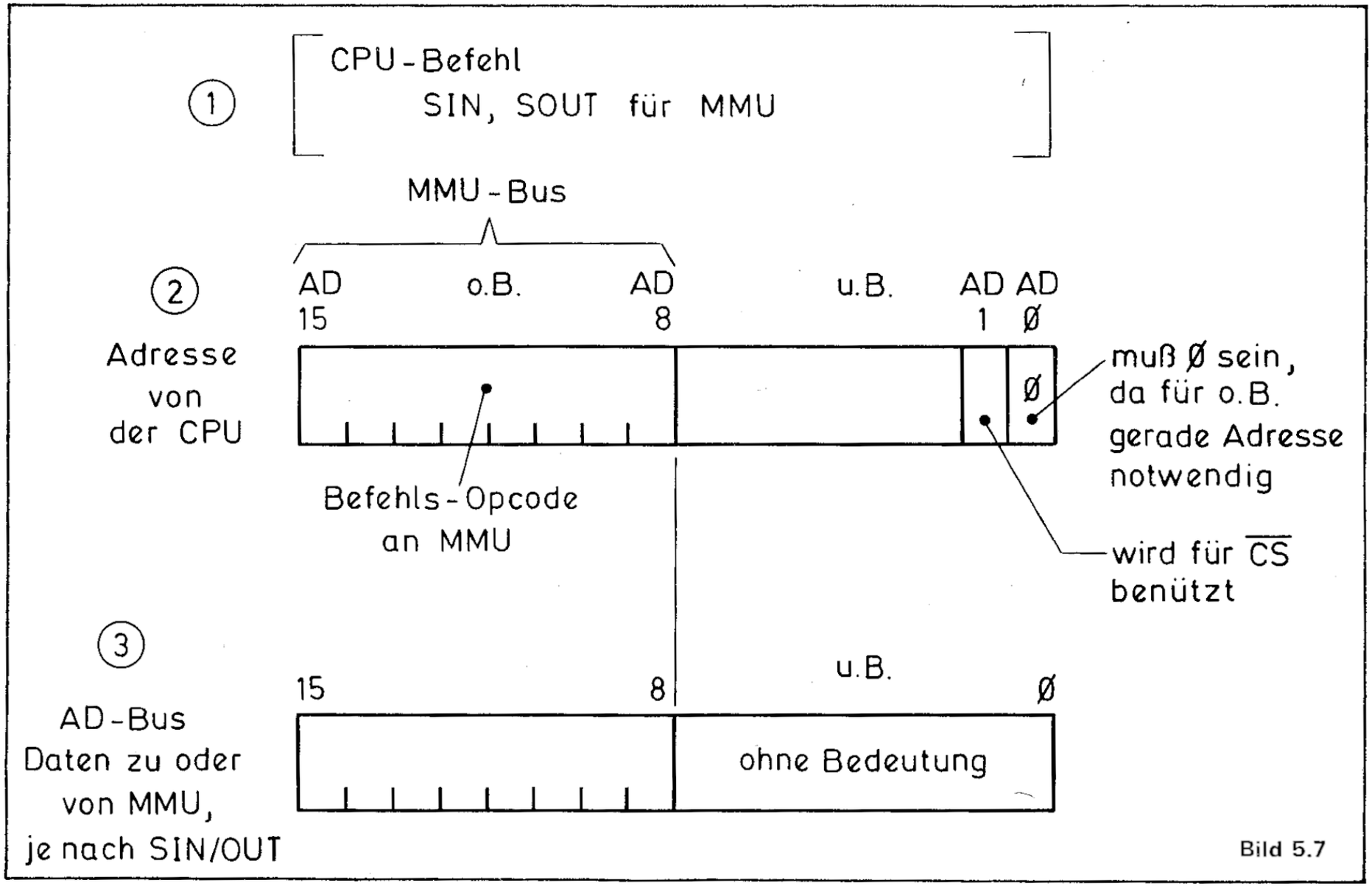


Bild 5.7

auch den CPU-Betriebszustand, während des unzulässigen Zugriffs, und zwar den CPU-Status ST0 . . . ST3, N/$\overline{S}$ und R/$\overline{W}$. In den MMU-Statusregistern verbirgt sich nach einem Trap an die CPU eine fast lückenlose Information des "warum, wieso und wo" des Traps, so daß eine Rekonstruierbarkeit des Ereignisses gegeben ist. Das 'fast lückenlos' bezieht sich auf den Ort des Speicherzugriffs. Der Offsetteil der Segmentadresse ist nur von A8 . . . A15 gespeichert. A0 . . . A7 der unzulässigen Zugriffsadresse muß in einen externen Zwischenspeicher abgelegt werden, wenn man das so haben will. Für eine Rekonstruktion des

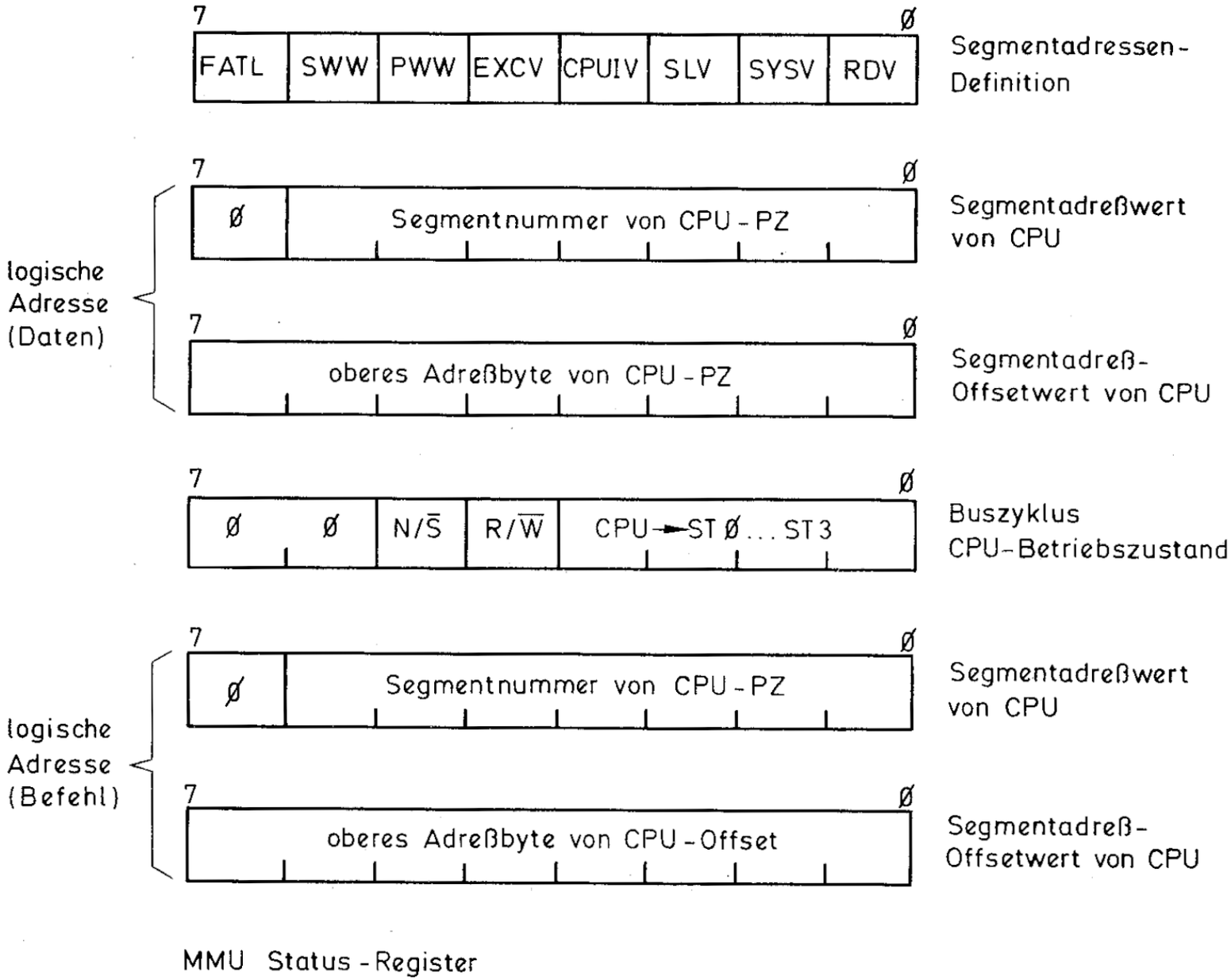

7
Ø
FATL
SWW
PWW
EXCV
CPUIV
SLV
SYSV
RDV
Segmentadressen-
Definition
logische
Adresse
(Daten)
Ø
Segmentnummer von CPU-PZ
Segmentadreßwert
von CPU
oberes Adreßbyte von CPU-PZ
Segmentadreß-
Offsetwert von CPU
Ø
N/S̄
R/W̄
CPU→ST Ø ... ST 3
Buszyklus
CPU-Betriebszustand
logische
Adresse
(Befehl)
oberes Adreßbyte von CPU-Offset
MMU Status-Register

Bild 5.8

Falles wäre dies jedoch nicht notwendig.

Zur Vollständigkeit werden kurz die Statusflags zur Anzeige eines unzulässigen Zugriffs vorgestellt (in steigender Stellenzahl).

RDV-Flag: Wird gesetzt, wenn die CPU in ein Segment schreiben will (R/$\overline{W}$ gleich Lo), das nur gelesen werden darf.

SYSV-Flag: Wird gesetzt, wenn die CPU im Normal-Modus in ein Segment zugreifen will, das nur im System-Modus aufgerufen werden darf.

SLV-Flag: Wird gesetzt, wenn die CPU eine Offsetadresse ausgibt, die außerhalb des Segments liegt.

CPUIV-Flag: Wird gesetzt, wenn die CPU in ein Segment zugreift und im Eigenschaftsfeld des zugehörigen Beschreibungsregisters das CPU-Inhibit-Flag gesetzt war.

EXCV-Flag: Wird gesetzt, wenn die CPU in ein reines Ausführungssegment auf eine andere Weise liest als im Rahmen eines regulären Befehlsholzyklus.

PWW-Flag: Wird gesetzt, wenn die CPU in den 256 Byte-Bund mit der Zahl N + 1 eines Segments schreibt (letzten 256 Byte des Segments), wobei das DIRW-Flag gesetzt war (primäre Schreibwartung).

SWW-Flag: Wird gesetzt, wenn die CPU im System-Modus Daten in den letzten 256 Byte-Bund (N + 1) eines Segments (als System-Stapel verwendet) schiebt und zugleich noch irgendeine andere Flag-Definition verletzt wurde wie RDV, SLV, EXCV und CPUIV. Wenn dieses Flag gesetzt wird, befindet sich der System-Stapel nahe am überlaufen. Ein Trap wird in diesem Fall jedoch nicht ausgegeben, nur $\overline{SUP}$ wird aktiviert.

FATL-Flag: Wird generell gesetzt, wenn ein Bedingungsflag betroffen bzw. verletzt wurde oder eine Schreibwarnung im Normalmodus entstanden ist. FATL wird nicht beim Laden des System-Stapels gesetzt, währenddessen eine Warnbedingung aufgetreten ist. Dieses Flag wird speziell dann aktiviert, wenn bei einem Speicherzugriff in einer Trap-Serviceroutine eine Verletzung einer Bedingung bzw. beim Zugriff ein Fehler aufgetreten ist. Bei irgendeiner Verletzung wird in der Trap-Serviceroutine kein weiterer Trap erzeugt; das wäre auch des Guten zuviel und vor allem nicht mehr sinnvoll. Es wird im gegebenen Fall nur das Suppress-Signal aktiviert, bis FATL auf Null gesetzt wurde.

Auf Grund der Bedeutung des Segmentadreß-Definitionsregisters ist es leicht denkbar, daß dieses Register mit einem Befehl auf Null zu setzen ist, um definierte Anfangsbedingungen zu schaffen. Das ist auch möglich und wird in der direkten Adressierungsart im SOUT-Befehl mit der Adresse % 1100 (zugleich Befehl im oberen Adreßbyte) ausgeführt. Ferner kann auch das SWW- und FATL-Flag einzeln auf Null gesetzt werden. Da mit den Befehlen die Initialisierung der MMU angesprochen ist, werden in der folgenden Auflistung alle MMU-Befehle vorgestellt. Die Segment-Beschreibungsregister werden bei einer Grundinitialisierung im Blockmodus geladen, und zwar von einem Initialisierungs-ROM direkt in die 64 Beschreibungsregister mit Hilfe des SOTIR-Befehls. Der Doppelwort-Befehl SOTIR ist in indirekter Adressierungsart für Quell- und Zielregister ausgelegt. Das Quellregister wird automatisch inkrementiert; im Zielregister wird in der E/A-Adresse der MMU zugleich auf AD8 . . . AD15 der MMU-Befehl plaziert. Für die automatische Inkremetierung der Segmentadresse wird % 0FXX (XX = gerade Adresse, dekodiert auf eine MMU) festgelegt.

Dieser Ein-Byte-Opcode eines MMU-Befehls ist an der höherwertigen Stelle (oberes Byte) der E/A-Adresse des je-

Befehle für den Zugriff auf die Segment-Beschreibungsregister

Opcode [1]	Befehl	Opcode	Befehl
08H	Basis-Feld	15H	alle CPU-Inhibitflags werden in das Segment-Beschreibungsregister gesetzt
09H	Limit-Feld		
0AH	Definitions-Feld		
0BH	Segment-Beschreibung (alle drei Felder)	16H	alle DMA-Inhibitflags werden in das Segment-Beschreibungsregister gesetzt
0CH	Basis-Feld, inkrementieren von SAR [2]		
0DH	Limit-Feld, inkrementieren von SAR		
0EH	Definitions-Feld, inkrementieren von SAR		
0FH	Segment-Beschreibung, inkrementieren von SAR		
0EH	Definitions-Feld, inkrementieren von SAR	1 :	diese Opcodes gelten für die jeweiligen Register zum Lesen und Schreiben (SIN, SOUT)
0FH	Segment-Beschreibung, inkrementieren von SAR	2 :	Segmentadreßregister

Befehle für den Zugriff auf die MMU-Steuerregister

Opcode [1]	Befehl	Opcode	Befehl
00H	Betriebsart-Register	1 :	diese Opcodes gelten für die jeweiligen Register zum Lesen und Schreiben (SIN, SOUT)
01H	Segmentadreßregister		
20H	DSC-Teil		

Befehle für den Zugriff auf die MMU-Statusregister

Opcode [1]	Befehl	Opcode	Befehl
02H	Segmentadreßdefinition	11H	Reset SAD [2]
03H	Segmentnummer bei Trap	13H	Reset SWW-Flag in SAD
04H	Segmentoffset (oberes Byte) bei Trap	14H	Reset FATL-Flag in SAD
05H	Buszyklus der CPU bei Trap	1 :	Statusregister nur lesen
06H	Segmentnummer bei Befehl	2 :	Segmentadreßdefinition
07H	Segmentoffset (oberes Byte) bei Befehl		

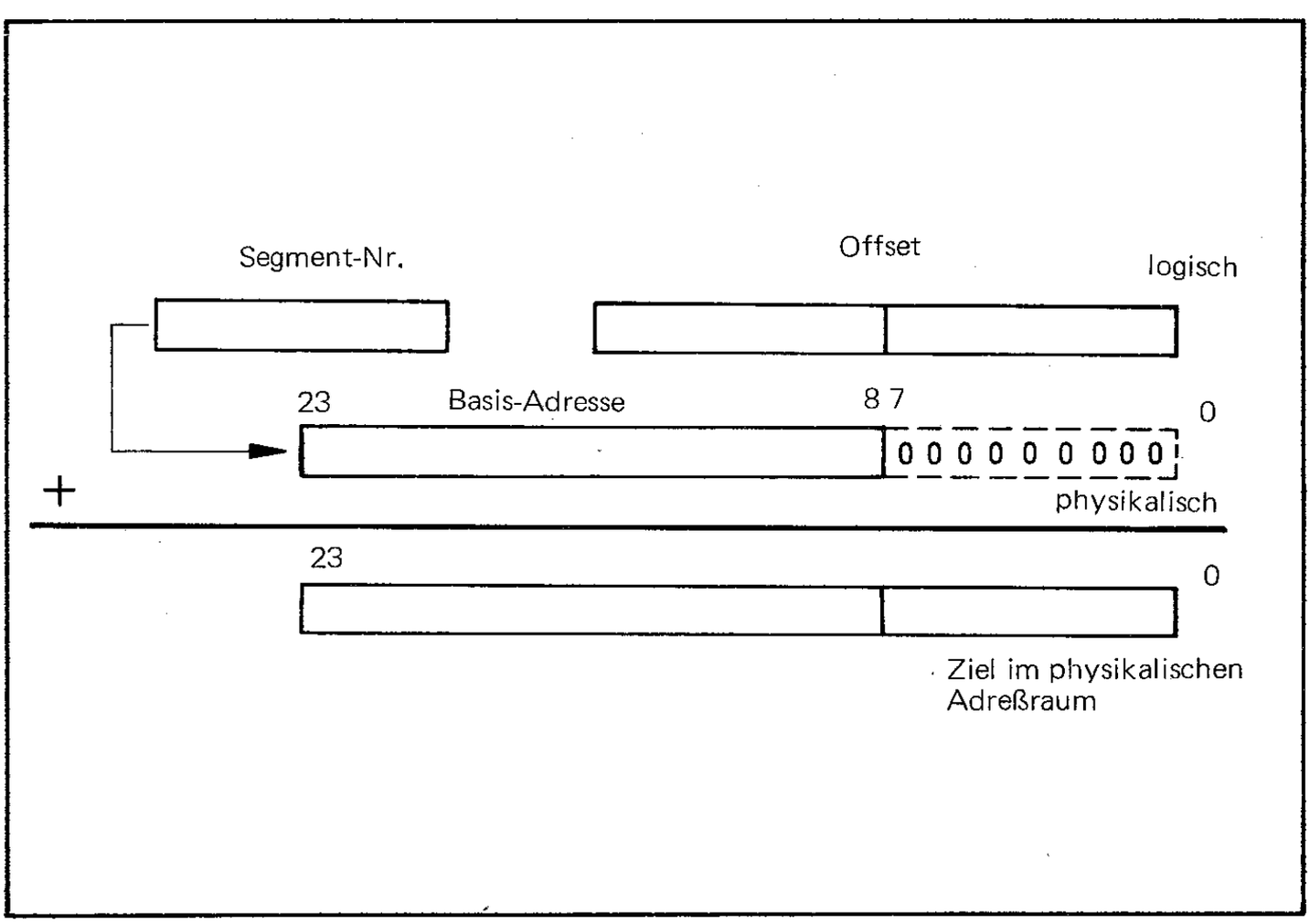
weiligen Spezial-E/A-Befehls einzubauen. Die Initialisierung gilt schon für eine MMU als relativ umfangreich. Dies muß man jedoch widerspruchslos hinnehmen, will man solch hohe Freiheitsgrade bei der Definition eines Segments beanspruchen. Baut man beispielsweise in ein Z 8001-System vier MMUs ein, zwei für 128 Normalmode-Segmente und zwei für 128 Systemmode-Segmente, benötigt man allein 1 KB Speicherkapazität nur für die Initialisierung der Segmentbeschreibungsregister. Rechnet man noch alle anderen CPU eigenen und E/A-Port bezogenen Initialisierungen hinzu, sind 2 KB als grobe Angabe für die Länge eines Initialisierungsprogramms keinesfalls zu hoch gegriffen. Es kommt hier sehr auf die Anwenderfreundlichkeit des Betriebssystems darauf an, woraus 2 KB durchaus überschritten werden können. Dieses Programm wird automatisch nach jedem Einschalten des Systems von der CPU ausgeführt.

Wichtig dabei ist, daß das Initialisierungsprogramm in einem ROM eingeschrieben und direkt von der Null-Seite startend zugriffsfähig ist (Beispiel nach Abschnitt 4.9). Über eine MMU darf die Adressierung nicht erfolgen.

Bild 5.9 zeigt einen möglichen Fall der MMU-Adreßausgabe nach entsprechender Initialisierung. Im letzten Segment (25FFH . . . 26FEH) wird ein Suppress-Signal aktiviert, wenn das DIRW-Flag gesetzt war. 26H steht im Limitfeld. An Stelle 26FFH kann wieder geschrieben werden, wenn andere Bedingungen das zulassen. In **Bild 5.10** ist ein Segment bis 25FFH beschrieben, wurde jedoch im Limitfeld mit FFH definiert. Man muß darauf achten, daß keine Programmadresse ab 2600H aufwärts vorkommt.

5.1.2 Definition des physikalischen Adreßfeldes

Physikalische Adresse, physikalischer Speicherraum sind Begriffe, die bereits aufgetreten sind. Der physikalische Speicherraum ist die jeweilige Menge von Hauptspeicheradressen in einem Z 8000-System mit MMUs, die effektiv belegbar sind. Die theoretisch belegbare Menge ist 16 MB. Die effektive Menge ist mit direkt adressierbarer Kapazität belegt. Der physikalische Adreßraum ist etwas reales. Das physikalische daran ist die Berechnung der effektiven Speicheradresse durch die MMU. Der hardwaretechnische Vorgang wurde in Bild 5.2 erläutert. Die Umsetzung von logischer Segment- und Offsetadresse in eine 24 bit lange Adresse ist das Charakteristikum daran.



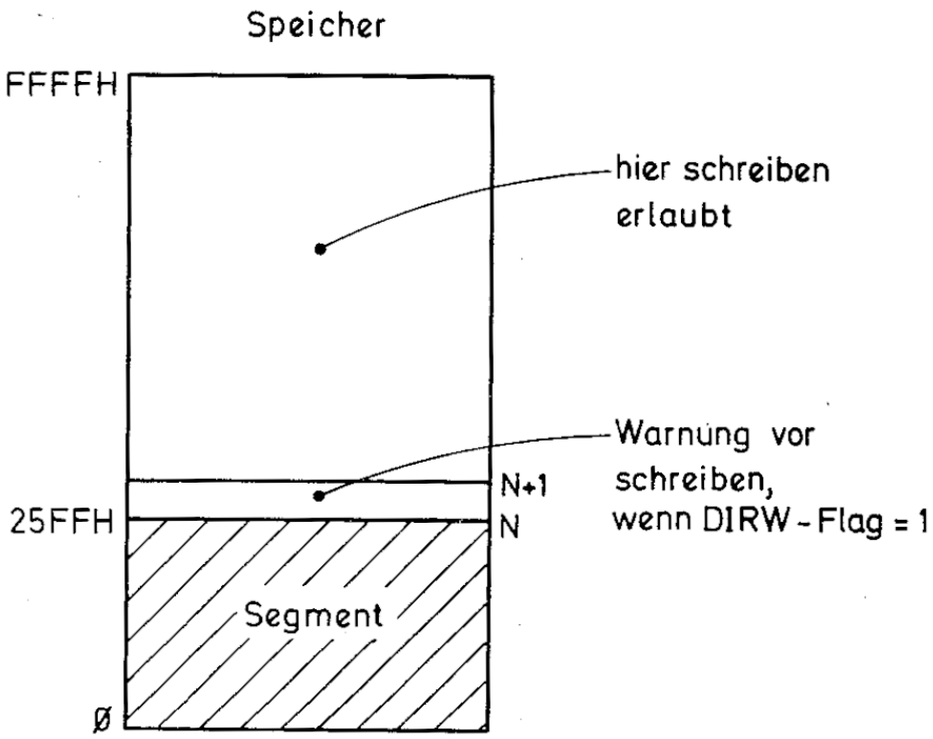


Bild 5.9

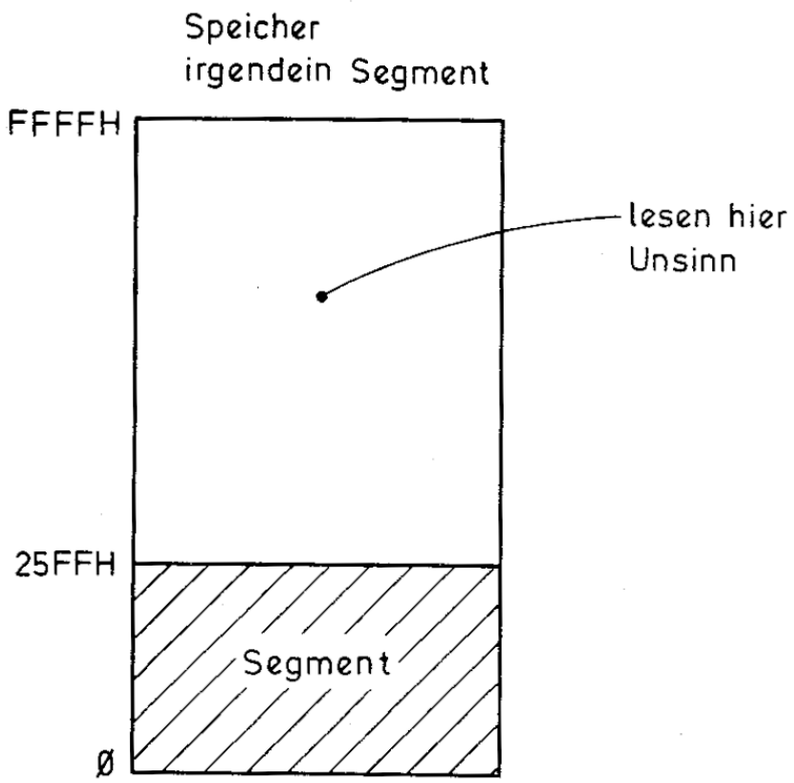


Bild 5.10

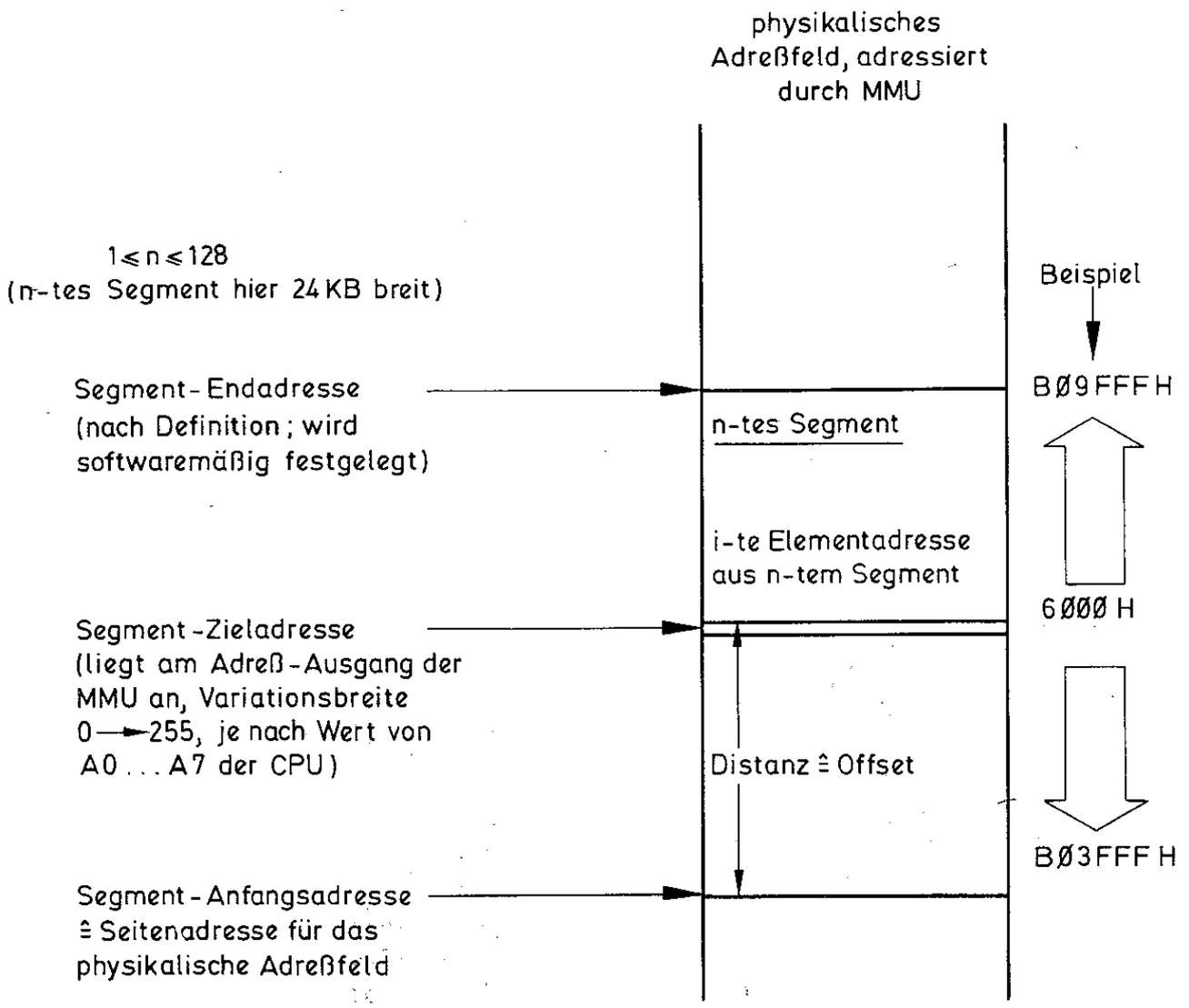


Bild 5.11

Die Basisadresse ist rechnerisch um 8 Stellen vom Komma nach links verschoben, beinhaltet deshalb von A0 . . . A7 immer Nullen.

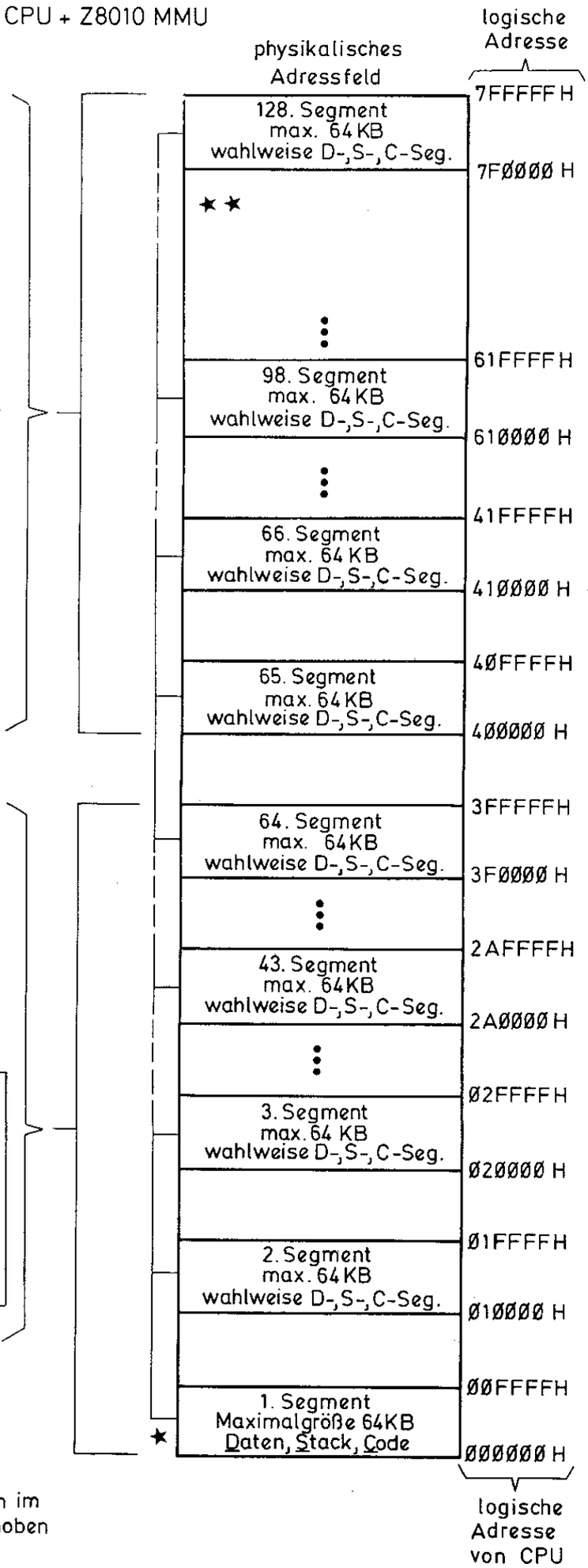
In **Bild 5.11** sind die ein Segment definierenden Adreßparameter dargestellt. Den Begriff Seitenadresse für ein Segment sollte man nur als Gedankenstütze verwenden, weil die Segmentanfangsadresse irgendwo im physikalischen Adreßraum liegen kann. Faßt man die 128 Segmente (bzw. 64 einer MMU) als Seiten auf, kann die Numerierung nach Wunsch erfolgen. Wenn man so will, kann die Seite 1 von der Seite 25 gefolgt sein, Seite 74 von Seite 12 . . .; dieser Freiheitsgrad gestattet beliebige Folgen. Der Z 8000 und die MMU beenden das klassische Konzept einer Seitenorientie-

rung. In **Bild 5.12** wird die Aufteilung der 128 Segmente nach dem strengen Schema der Reihenfolge dargestellt. Die logische Adresse gibt den Bereich an, von 0H . . . 0FFFFH das erste, von 10000H . . . 1FFFFH das zweite und von 7F0000H . . . 7FFFFFH das letzte Segment. Eine MMU wird auf die Segmente 65 . . . 128 festgelegt. Dabei ist erforderlich, daß das URS-Flag im Betriebsart-Register auf 1 gesetzt wird. Die zweite MMU wird auf die Segmente 1 . . . 64 festgelegt (URS-Flag gleich Null). Würde man einen Übersetzungsfaktor von 2 wählen (festzulegen im Basis-Feld des Beschreibungsregisters), wäre die erste Adresse auf 0H zu legen, die zweite auf 20000H, die dritte auf 40000H . . . die letzte auf FF0000H. Sicherlich wäre das unsinning. Das Beschreibungsfeld gestattet jedoch diese Definition. Eine Verteilung dieser Art ist deshalb unsinnig, da einerseits Speicherplatz verschenkt wird und andererseits (was viel wichtiger ist) die Zuordnung von Segmenten nicht ausgenutzt wird. Es läßt sich an dieser Stelle keine feste Regel einer Zuordnung vorgeben, weil es sie nicht gibt. Jeder Anwender muß sich die Zuordnung selbst ausdenken. **Bild 5.13** zielt genauer darauf ab, was damit gemeint ist. Die Verteilung der Segmente läßt sich auf gleiche Ebene übertragen, gegeneinander überlappen, oder aneinander angrenzen.

Für eine dynamische Segmentzuordnung gibt es eine Reihe von Gründen. Ein bestimmtes Programmcode-Segment soll mit vielerlei Datenquellen (Segmenten) in Zusammenhang gebracht werden, d.h. die Datenmanipulation wird nach einem bestimmten Schema vorgenommen. Es könnten mehrere Segmente mit einem Anwenderprogramm in einer bestimmten Sprache anstehen, die vom Anwender einen Bedarf auf die Übersetzung in den Maschinencode anmelden (Compiler-Segment-Anforderung). Ebenso kann eine bestimmte Textquelle (Anwenderprogramm) sowohl für eine Codeumwandlung allgemeiner Art als auch zu einer Compilierung anstehen. Alle diese verschiedenen Bereiche spielen sich auf verschiedenen Segmentebenen ab, gleichgültig, ob daran ein oder mehrere Anwender beteiligt sind. Aus der vielfältigen Organisation von Segmenten verschiedener softwaremäßiger Bedeutung ist für einen Anwender beispielsweise die Art nach **Bild 5.14** möglich. Zwei Programmcode-Segmente mit je 2 KB und 16 KB werden von zwei Stapelbereichen unterstützt. Das Datenbereich-Segment Nr. 75 kann nur von dem Anwender selbst benutzt werden. Der Anwender startet aus seinem Codesegment 2 (Segment 82) einen Zugriff in den Datenbereich, indem er "CALL 75, INTEGER" benennt. Nach entsprechendem Programm wird dieses Label von der betroffenen MMU in die entsprechende physikalische Adresse übersetzt. Würde in den oberen 256 Byte-Bund des Stack-Bereichs 2 geschrieben werden, würde ein Trap erzeugt. Diese als Schreibwarnung zu verstehende Unterbrechung soll dem Anwender andeuten, daß dieser Stapelbereich mehr Speicherraum benötigt. Ein Suppress-Signal wird nicht erzeugt, damit die Stapeloperation nicht behindert wird.

Bild 5.15 zeigt die Zuordnung jeweils dreier Segmente (Code, Daten und Stapel) für zwei Anwender. Die Einteilung ist hier noch ohne Überlappung angegeben. In **Bild 5.16** ist eine mögliche Überlappung dargestellt, wie sie in Bild 5.13 angedeutet wurde. Generell ist sie für alle drei Arten eines Segments, Programm-, Daten- und Stacksegment möglich. Die Zuordnung der Basisadressen ist geeignet zu ermitteln. Werden zu einem Segment bestimmte Bedingungen im Eigenschaftsfeld, z.B. für Segment 2 definiert, gelten diese selbstverständlich nicht für das Segment 3 innerhalb des gleichen Code-Bereiches. Es gelten nur die Bedingungen des Segments 2, in dessen Rahmen man im physikalischen Speicherraum zugreift. **Bild 5.17** zeigt den Grund einer Überlappung bis zur vollständigen Deckung physikalischer Speicheradressen beider Segmente. Zwei Anwender benutzen bei ihrer Programm-

16 MByte Adreßfeld ; Z8001 CPU + Z8010 MMU



1. Z8010 MMU

(Speicherorganisationsbaustein)
erfaßt die Segmentnummern
65 → 128
wenn URS-Flag = 1 ;
URS im Betriebsartsteuer-register legt den Seg.-Bereich fest.
Im Steuerregister "Segmentbeschreibungsadresse" steht die Zahl des Segments des zu adressierenden Segmentnummernintervalls.
Im "Limit"-Feld des Segmentbeschreibungsreg. ist die Segmentgröße
256 Byte → 64 KByte
festgelegt.

2. Z8010 MMU

erfaßt die Segmentnummern
1 → 64
wenn URS-Flag = 0 ;

Für 128 Segmente
(von Z8001 CPU adressierbar)
sind
zwei MMUs
notwendig

★ Z8002 CPU kann nur ein Segment adressieren

★★ Jedes Segment kann um (16 MB - Segmentbreite) dynamisch im physikalischen Adressfeld verschoben werden.

Bild 5.12

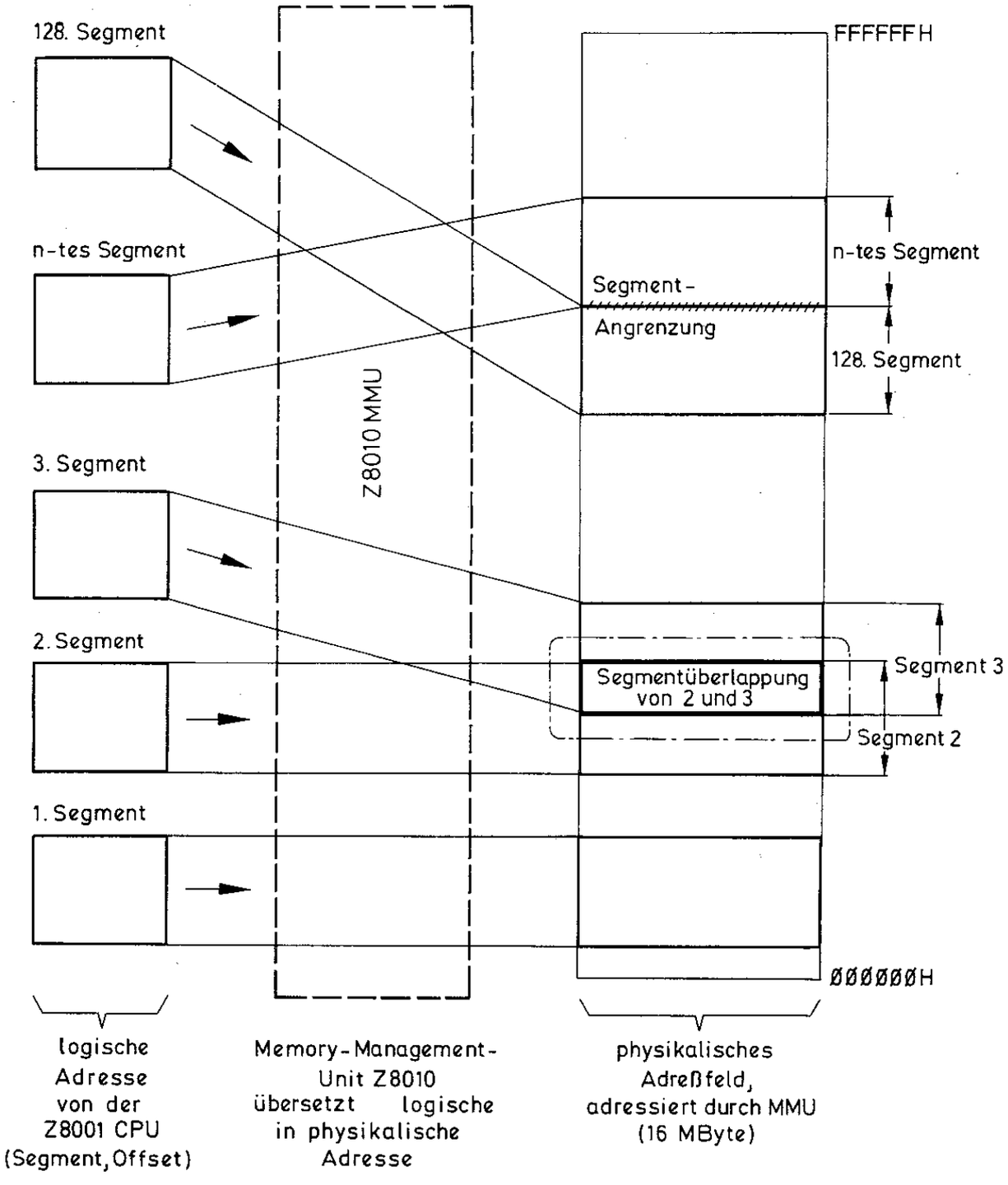

128. Segment
n-tes Segment
3. Segment
2. Segment
1. Segment
Z8010 MMU
FFFFFFH
Segment-
Angrenzung
n-tes Segment
128. Segment
Segmentüberlappung
von 2 und 3
Segment 3
Segment 2
ØØØØØØH
logische
Adresse
von der
Z8001 CPU
(Segment, Offset)
Memory-Management-
Unit Z8010
übersetzt logische
in physikalische
Adresse
physikalisches
Adreßfeld,
adressiert durch MMU
(16 MByte)

Bild 5.13

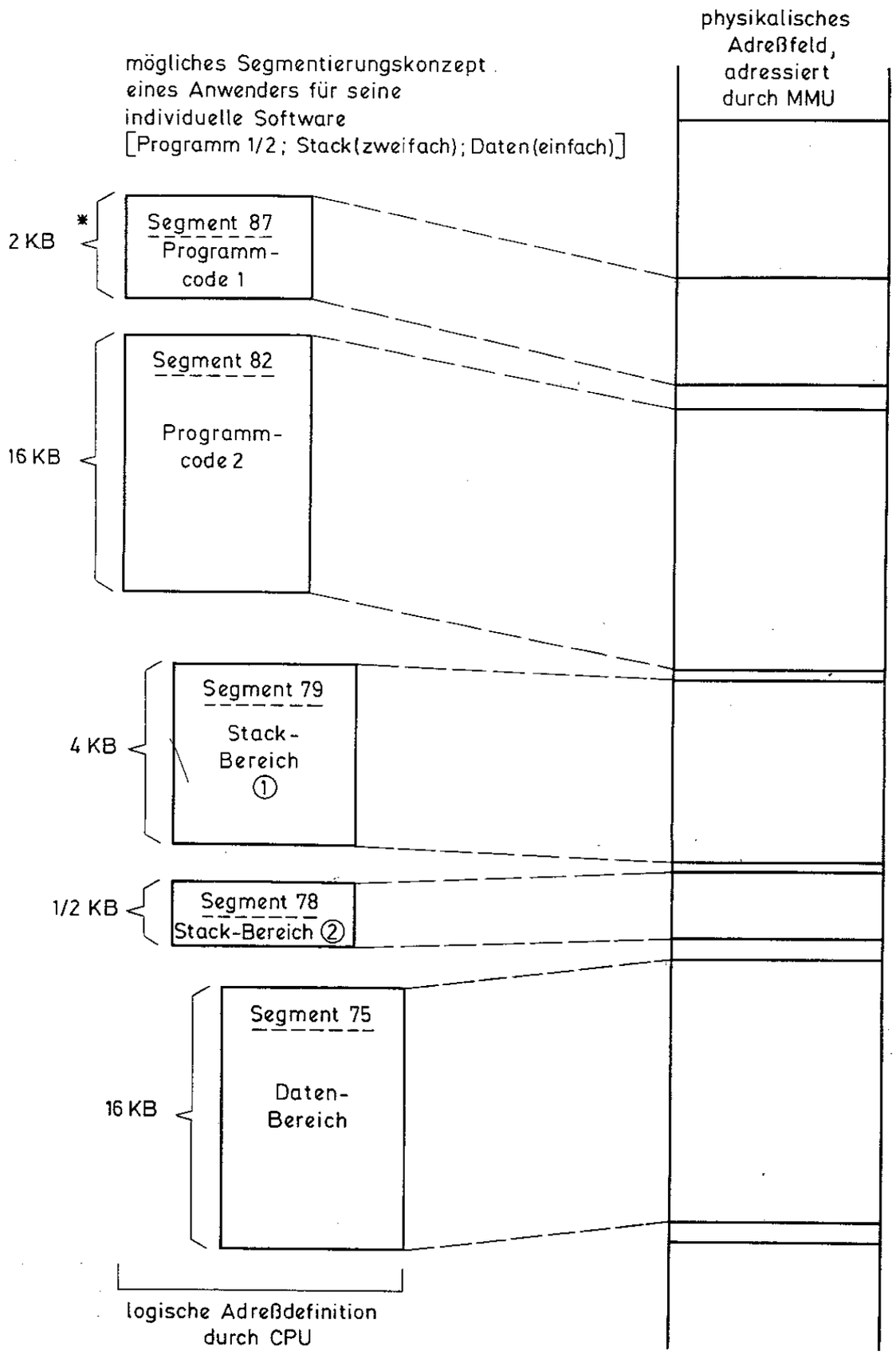

mögliches Segmentierungskonzept
eines Anwenders für seine
individuelle Software
[Programm 1/2; Stack(zweifach); Daten(einfach)]
physikalisches
Adreßfeld,
adressiert
durch MMU
*
2 KB
Segment 87
Programm-
code 1
Segment 82
16 KB
Programm-
code 2
Segment 79
4 KB
Stack-
Bereich
①
1/2 KB
Segment 78
Stack-Bereich ②
Segment 75
16 KB
Daten-
Bereich
logische Adreßdefinition
durch CPU
* alle Angaben der Segmentbreite
sind beliebig gewählt
(256 Byte → 64 KB)

Bild 5.14

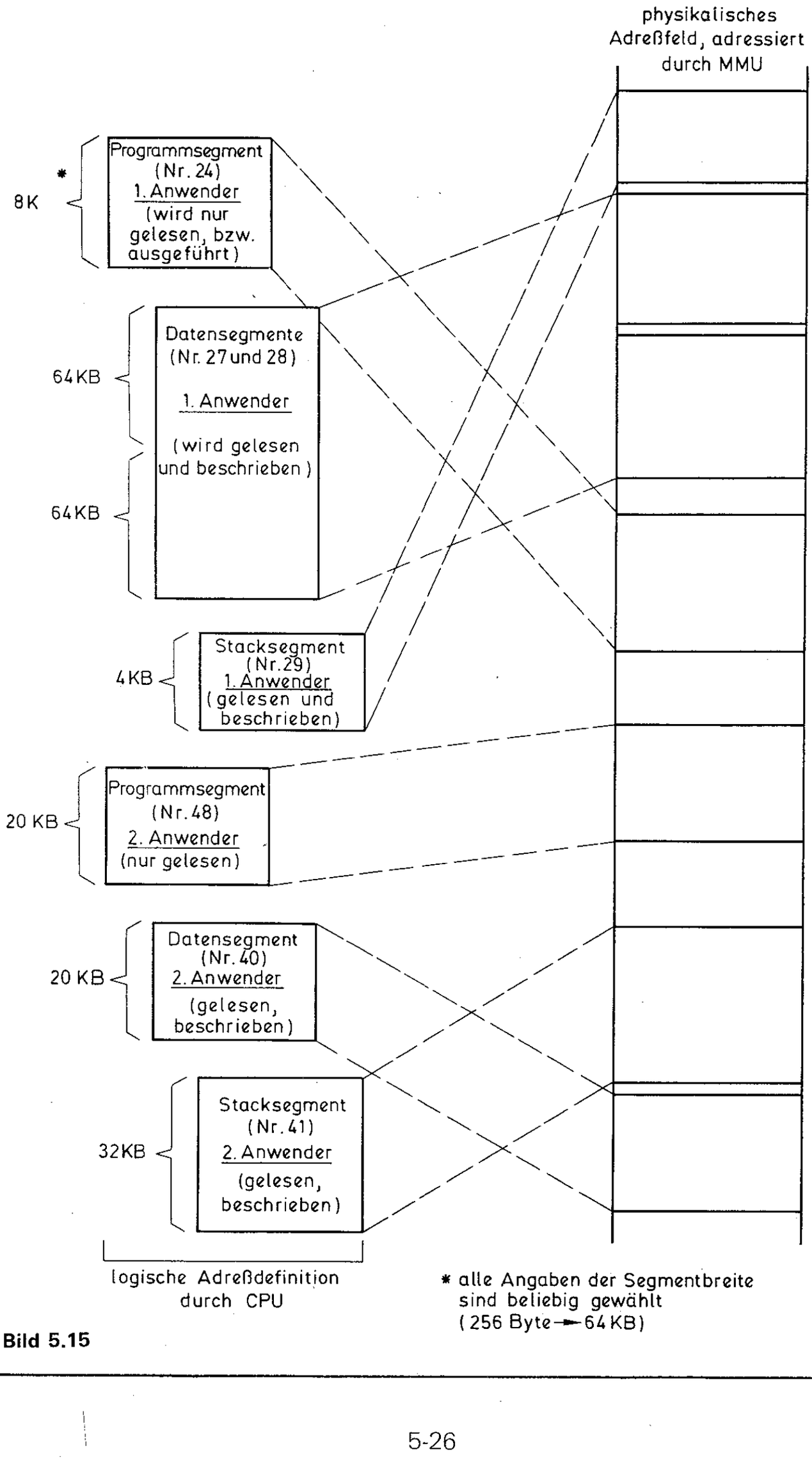

physikalisches
Adreßfeld, adressiert
durch MMU
*
8K
Programmsegment
(Nr. 24)
1. Anwender
(wird nur
gelesen, bzw.
ausgeführt)
64KB
Datensegmente
(Nr. 27 und 28)
1. Anwender
(wird gelesen
und beschrieben)
64KB
4KB
Stacksegment
(Nr. 29)
1. Anwender
(gelesen und
beschrieben)
20 KB
Programmsegment
(Nr. 48)
2. Anwender
(nur gelesen)
20 KB
Datensegment
(Nr. 40)
2. Anwender
(gelesen,
beschrieben)
32KB
Stacksegment
(Nr. 41)
2. Anwender
(gelesen,
beschrieben)
logische Adreßdefinition
durch CPU
* alle Angaben der Segmentbreite
sind beliebig gewählt
(256 Byte → 64 KB)

Bild 5.15

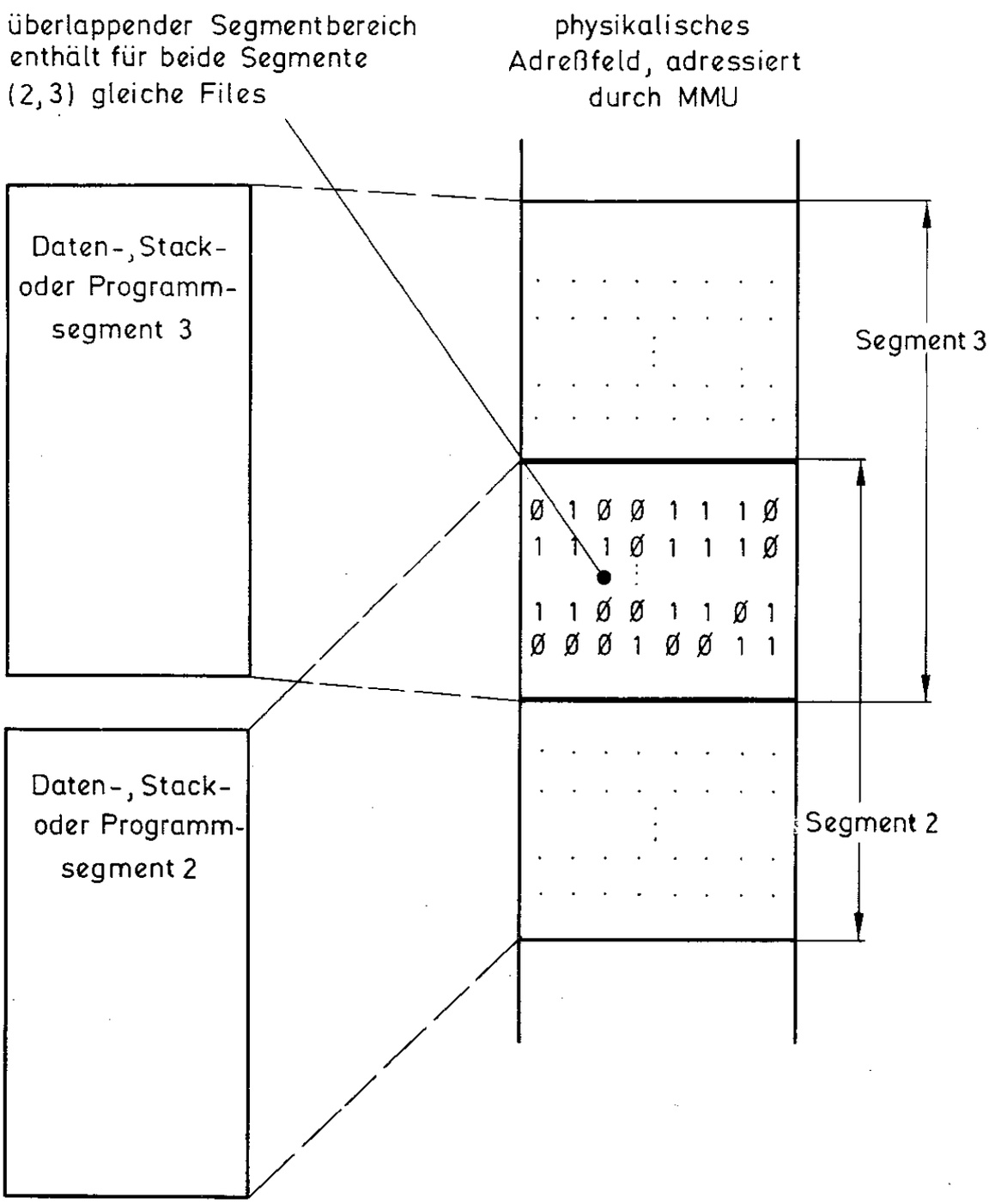

überlappender Segmentbereich
enthält für beide Segmente
(2,3) gleiche Files
physikalisches
Adreßfeld, adressiert
durch MMU
Daten-, Stack-
oder Programm-
segment 3
Daten-, Stack-
oder Programm-
segment 2
Segment 3
Segment 2

Bild 5.16

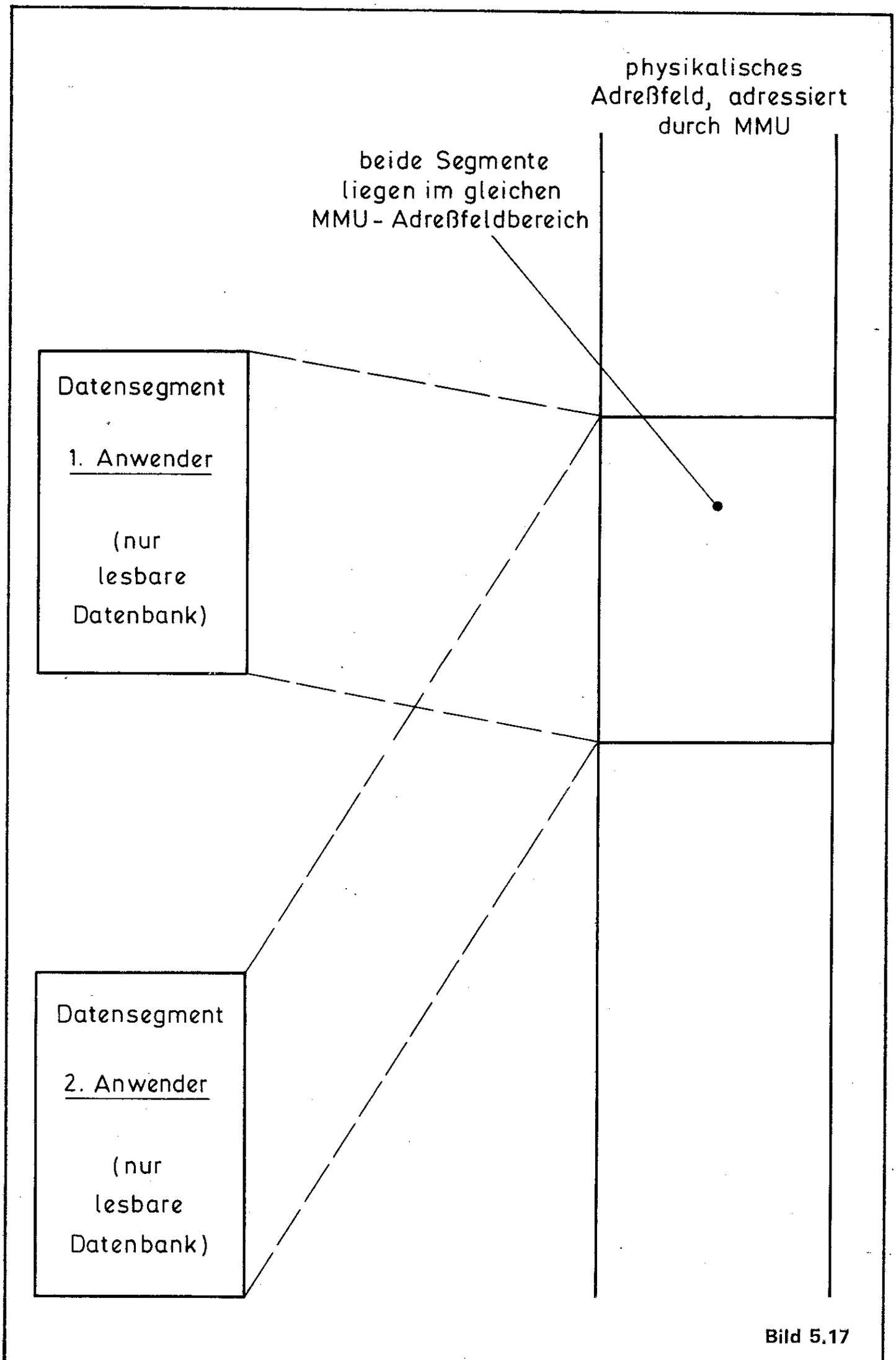

physikalisches
Adreßfeld, adressiert
durch MMU
beide Segmente
liegen im gleichen
MMU - Adreßfeldbereich
Datensegment
1. Anwender
(nur
lesbare
Datenbank)
Datensegment
2. Anwender
(nur
lesbare
Datenbank)

Bild 5.17

ausführung die gleiche Datei.
Die bisher gezeigten Varianten einer Segmentzuordnung zeigten die Gegebenheit einer Mehrbenutzer-Organisation und damit die generelle Zielsetzung der Kombination CPU und MMU. Die Mehrbenutzer-Organisation beinhaltet ebenso das virtuelle Speicherkonzept, wofür die Basis im Betriebssystem gegeben sein muß. Das Charakteristikum des virtuellen Speicherkonzepts besteht in zwei Punkten:

- Die Benutzeradressen für einen Anwender werden als Elemente von den maximal möglichen 16 MB Hauptspeicheradressen als unabhängige logische, virtuelle Adreßeinheit betrachtet. Je nachdem, wie viele Anwender sich den Rechner teilen, ist eine weite Grenze dieser virtuellen Adressenzahl denkbar. Es hängt davon ab, wie weit der 16 MB-Hauptspeicheradreßraum ausgebaut ist.
- Die Verknüpfung und Zuordnung zwischen Hauptspeicheradressen und virtuellen Adressen wird mit Hilfe der Adreßübersetzung der MMU hergestellt. Eine virtuelle Adresse wird aus der Hauptspeicher-Quelladresse berechnet. Der virtuelle Adreßraum muß praktisch aus dem Hauptspeicheradreßraum herausgelöst gedacht sein. Die Anzahl virtueller Speicherbereiche sind zahlenmäßig für die mögliche Anzahl der Anwenderplätze reserviert. Innerhalb eines Bereichs variabler Größe besteht freie Verfügbarkeit eines Anwenders.

Die im virtuellen Speicher fixierte Information wird durch Verwaltungsroutinen des Betriebssystems über verschiedene Speichermedien verteilt (externe Massenspeicher). Ein Segment-Trap der MMU an die CPU signalisiert beispielsweise die genaue Abgrenzung des virtuellen Speicherbereichs eines Anwenders, der über mehrere Segmente hinweg zu quasi einem Block zusammengefügt wird. Die Traps dienen sowohl zur Abgrenzung für den Anwender selbst als auch für andere Anwender, die in den jeweilig bestimmten virtuellen Bereichen "nichts zu suchen" haben. Das "wie" wird wiederum in den Eigenschaftsfeldern der Segment-Beschreibungsregister festgelegt.

5.1.3 Signalverhalten der MMU

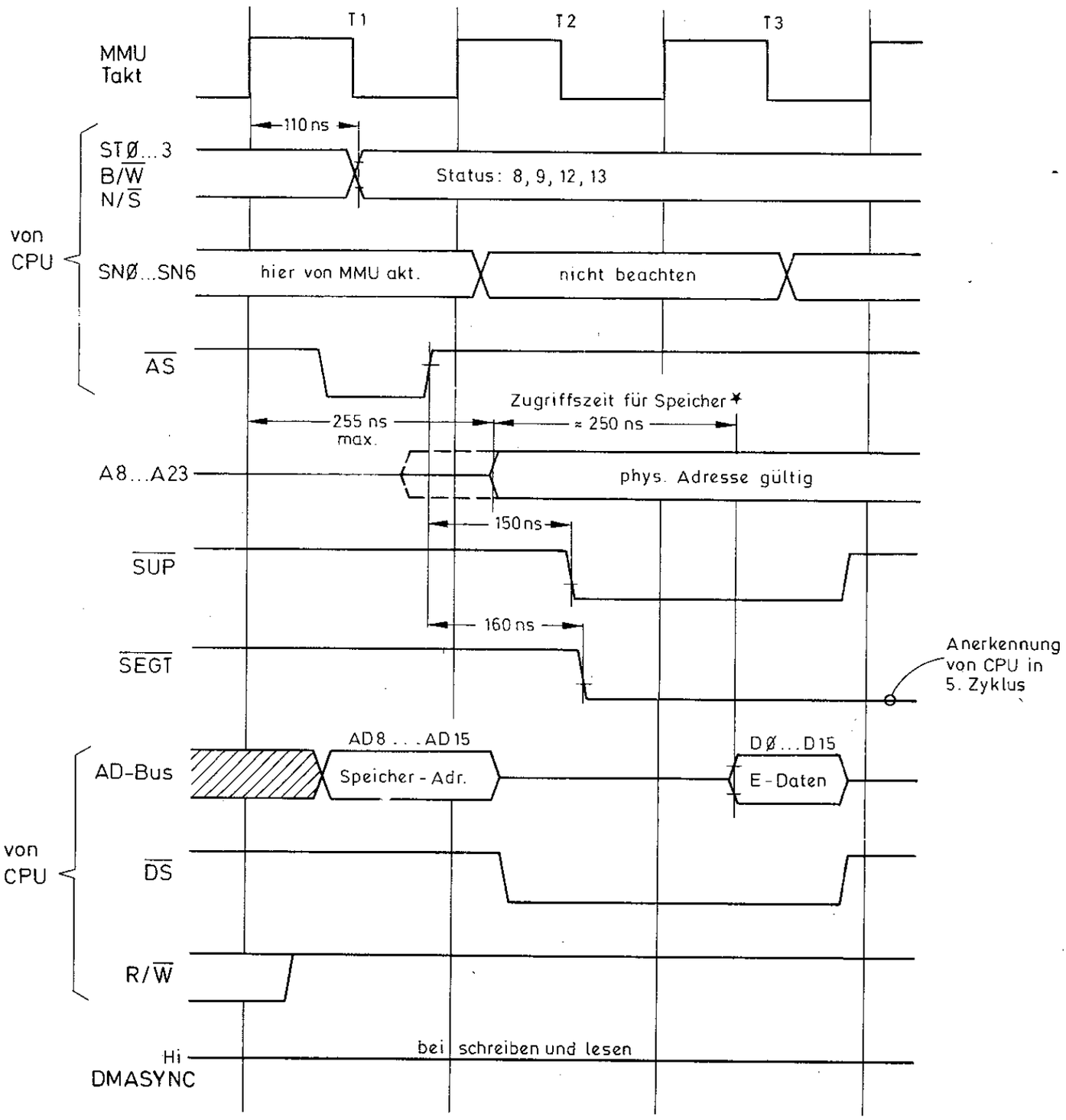
Die Z 8010-MMU ist auf die Taktzykluszeiten der Z 8001-CPU abgestimmt. Die Z 8001A-CPU erfordert die entsprechend schnellere Version der MMU. Wie bereits genannt wurde, ist für die 4 MHz Taktversion mit Speichern einer Zugriffszeit von 150 ns zu arbeiten, will man ohne Wartezyklus auskommen. Rein rechnerisch stehen bei einem Betrieb mit 250 ns Taktzyklus etwa 230 ns Zugriffszeit für die CPU zur Verfügung. Hier muß die Verzögerung bei der Dekodierung mit eingerechnet werden. Für die Kombination Z 8001A und Z 8010A kann zum Zeitpunkt des Erscheinens dieses Buches noch keine genaue Angabe gemacht werden.

Für das Signalverhalten der MMU sind im wesentlichen vier Diagramme maßgebend, die mit Speicher lesen, Speicher schreiben, Initialisierung der MMU im Rahmen eines speziellen E/A-Zyklus und dem Segment-Trap-Anerkennungszyklus zu bezeichnen sind.

Speicher lesen mit der MMU:

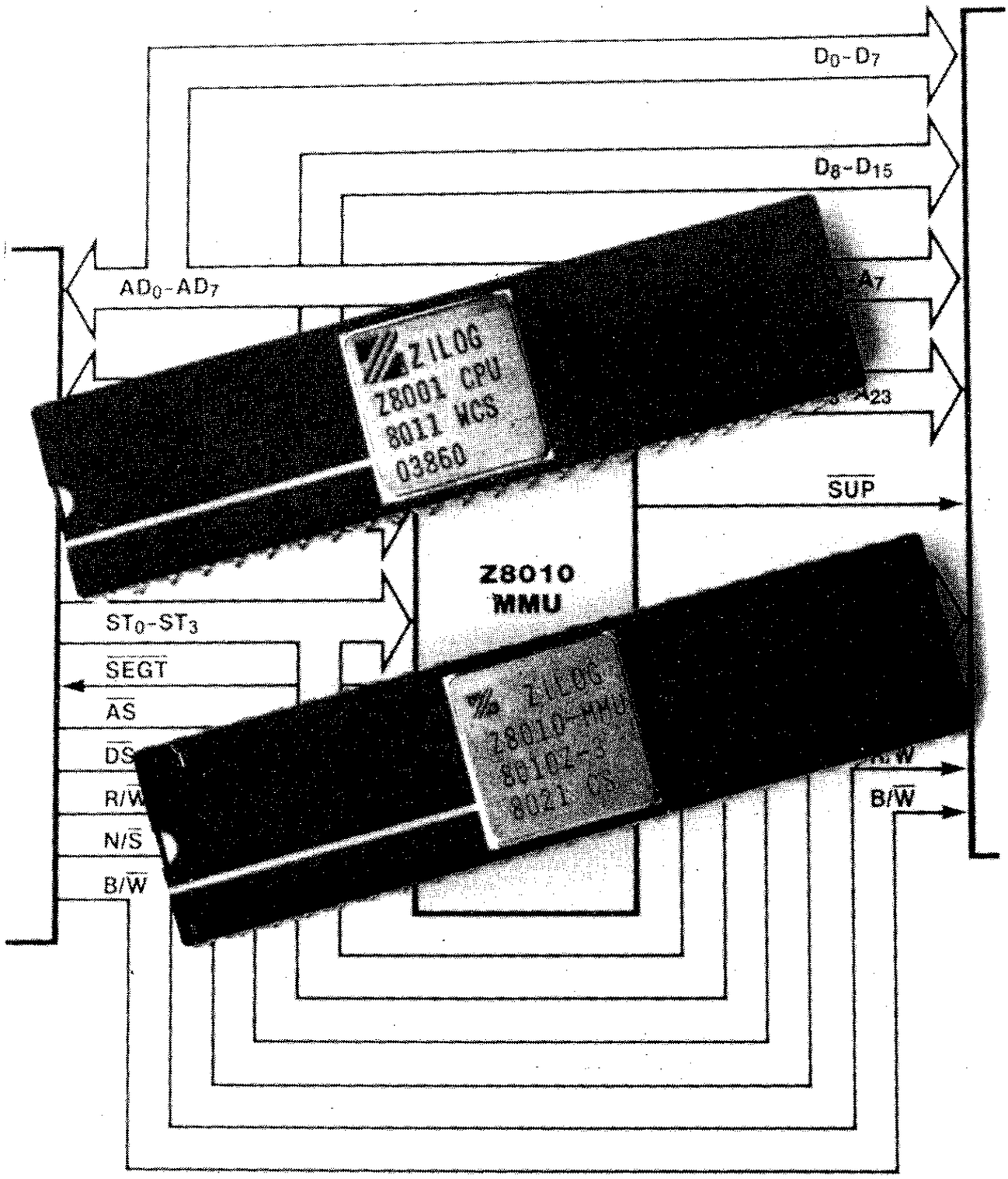
Bild 5.18 zeigt den Lesezyklus über die MMU mit dem Speicherwerk. Ein allgemeiner Lesezyklus unterscheidet sich von einem Befehlsholzyklus nur durch die Statusausgabe Nr. 12 bzw. Nr. 13. Legt die CPU eine Adresse auf den Bus, wird die Adreßumsetzung nach dem beschriebenen Verfahren durchgeführt, gleichgültig wie $\overline{CS}$ der MMU belegt ist. **$\overline{CS}$ gleich Lo ist nur erforderlich, wenn**

Speicher lesen mit MMU



* bis 250 ns Zugriffszeit des Speichers, -kein WZ für Speicher nötig (gegenüber 350 ns ohne MMU)

Bild 5.18


D0-D7
D8-D15
AD0-AD7
A7
A23
ZILOG
Z8001 CPU
8011 MCS
03860
SUP
Z8010
MMU
ST0-ST3
SEGT
AS
DS
R/W
N/S
B/W
ZILOG
Z8010-MMU
8021 CS

Speicher schreiben mit MMU

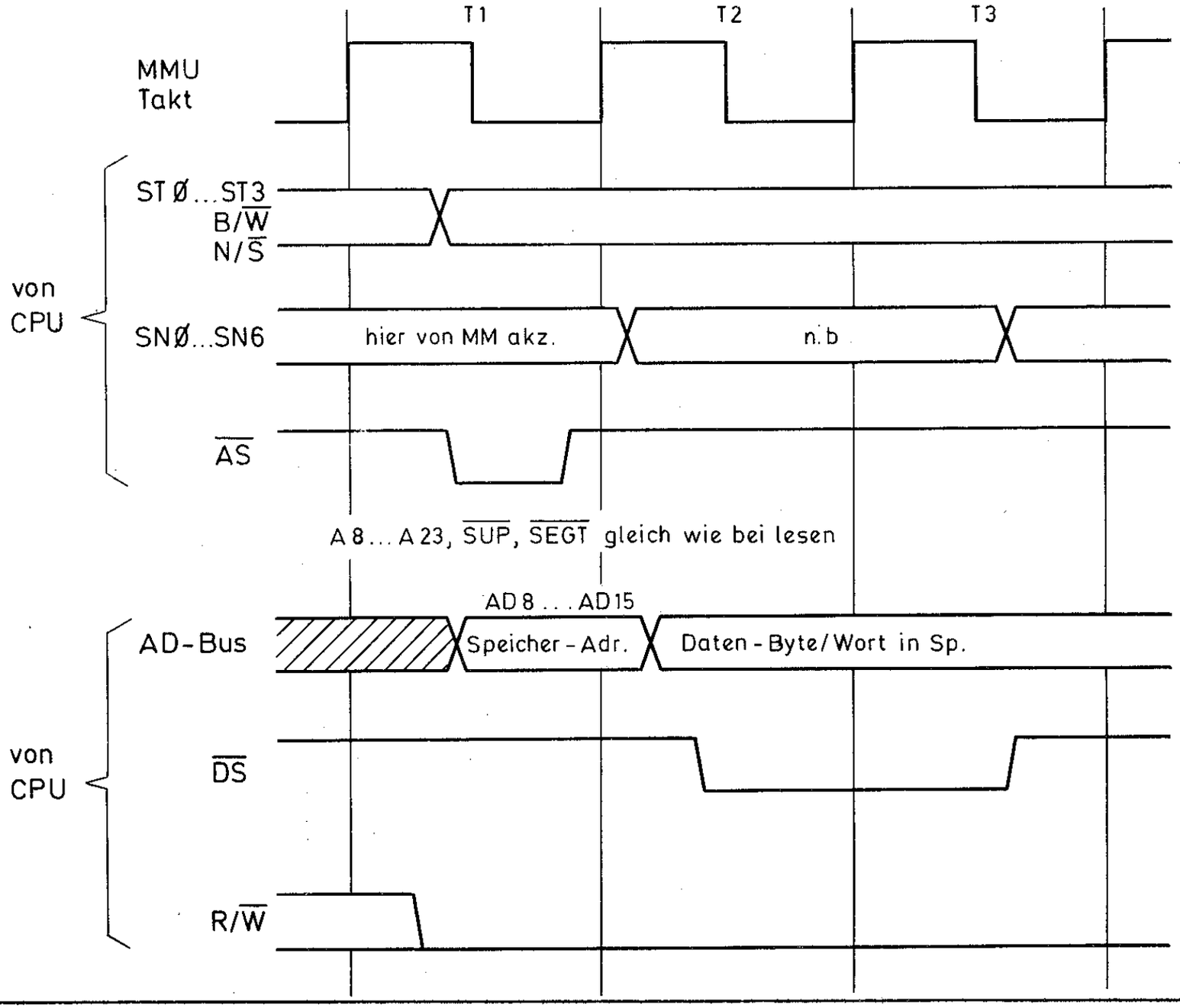


Bild 5.19

CPU - Steueroperation von und zu MMU

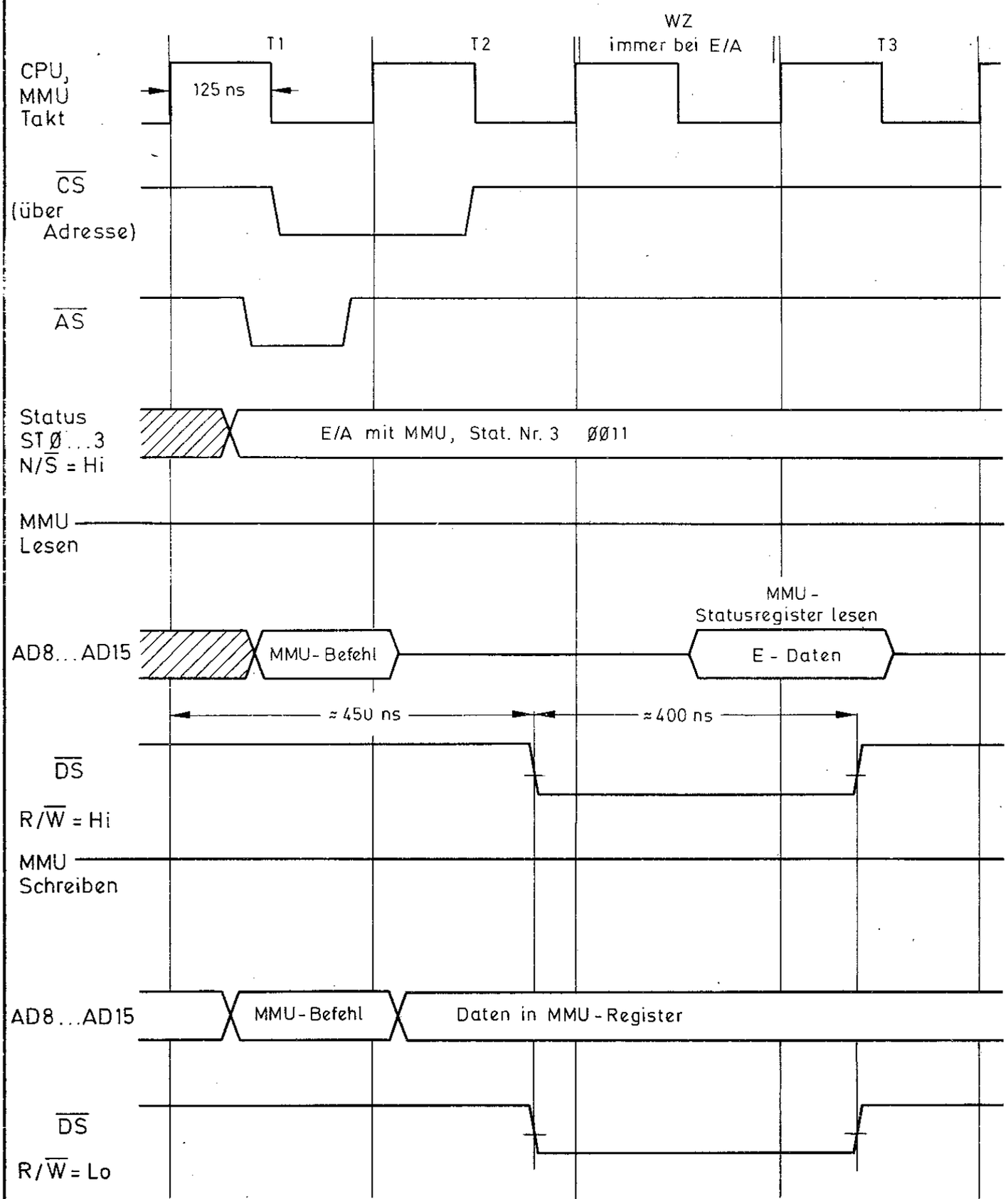


Bild 5.20

die MMU initialisiert bzw. ihre internen Register gelesen werden. Die Segmentadresse SN0 . . . SN6 wird im ersten Taktzyklus übernommen. Nach spätestens 255 ns ab Beginn von T1 wird die physikalische Adresse A8 . . . A23 ausgegeben. 250 ns sei als äußerste Grenze für einen Speicherzugriff ohne Wz angegeben. 250 ns Zugriffszeit des Speichers reicht in jedem Fall nicht mehr aus. Ein Weg ist, mit dem Taktzyklus länger zu werden und testen. Der MMU-Eingang DMASYNC muß Hi sein, wenn die Adresse von der CPU stammt. Die MMU behandelt intern das obere Byte der Offsetadresse an AD8 . . . AD15 generell an der Lo ↑ Hi-Flanke von $\overline{AS}$ als gültig. Die physikalische Adresse der MMU bleibt bis zum Ende von T3 eines Maschinenzyklusses gültig. $\overline{SUP}$ sowie $\overline{SEGT}$ sind etwa zur Hi ↓ Lo-Flanke von T2 aktiv, wenn sich aus dem internen Test ein Erfordernis ergibt.

Speicher schreiben mit der MMU:

Bild 5.19 zeigt den Schreibzyklus über die MMU mit dem Speicherwerk. Da die MMU in ihrem Verhalten nicht weiter betroffen ist, ob geschrieben oder gelesen wird, sind weitere Angaben hierzu überflüssig.

E/A-Zyklus zur Initialisierung der MMU:

Bild 5.20 zeigt diesen Zyklus, der die MMU direkt betrifft. Sollen die MMU-Register gelesen oder geladen werden, ist dies mit Hilfe der speziellen MMU-E/A-Befehle der CPU softwaremäßig einzuleiten. Den dabei von der CPU ausgegebenen Status Nr. 3 benützt die MMU zur internen Umschaltung und betrachtet die gleichzeitig ausgegebene E/A-Adresse der CPU am oberen Byte als Befehl. Ein Byte-Transfer an AD8 . . . AD15 schließt den Zyklus ab, der immer um einen Wartezyklus erweitert wird.

Segment-Trap-Anerkennungszyklus:

Bild 5.21 zeigt den Segment-Trap-Anerkennungszyklus der CPU, wie er von der MMU aus gesehen, sich auswirkt. Bei einer Verletzung der Bedingung eines Segments aktiviert die MMU auch den $\overline{SUP}$-Ausgang. Sind mehrere MMUs im System integriert, sendet diejenige, die den Trap ausgelöst hat, an ihren Ausgängen AD8 . . . AD15 ihre Kennung zu CPU, die im Identifikationsfeld 0 . . . 2 des Betriebsart-Registers festgelegt wurde. Alle anderen AD-Leitungen der übrigen MMUs sind während der Ausgabe der Kennung im hochohmigen Zustand.

Das Signalverhalten der MMU betrifft auch den Fall eines allgemeinen E/A-Zyklus (Statusausgabe Nr. 2). Die Konzeption der MMU beinhaltet keine Übersetzung einer allgemeinen E/A-Adresse für einen Schnittstellenbaustein. Das wäre auch höchst unsinnig. In diesem Fall werden die Anschlüsse AD8 . . . AD15 bzw. A8 . . . A23 in den hochohmigen Zustand versetzt. In dem gleichen Zustand verharren diese MMU-Leitungen bei einem Refresh-Zyklus der CPU. Auffrischadressen betreffen das untere Byte des AD-Busses, der an der MMU nicht anliegt. Die Darstellung der genannten Betriebszustände ist überflüssig.

5.2 ANSCHLUSSBILD DER MMU

Die Z 8010-MMU bildet mit der CPU den Kern eines 16-bit-Computers absolut höherer Ordnung. Bei der Entwicklung der MMU ist eine sehr enge Zusammenarbeit mit der Entwicklungsgruppe der CPU erfolgt. An der Entwicklung war maßgeblich ein zweiter Japaner beteiligt. Neben Masatoshi Shima für die CPU, war für die MMU Hiroshi Yonezawa mit dem Design betraut. Es ist sicherlich leicht denkbar, daß sich CPU und MMU nicht nur softwaremäßig ergänzen, sondern sich auch hardwaremäßig sehr gut vertragen. Die Verknüpfung zweier MMUs mit der CPU erfordert praktisch so gut wie keine zusätzliche Hardware.

Nachdem die MMU bereits gut genug

Segmentübertritt - Anerkennung von CPU; $\overline{SEGT}$ - Interrupt löst aus

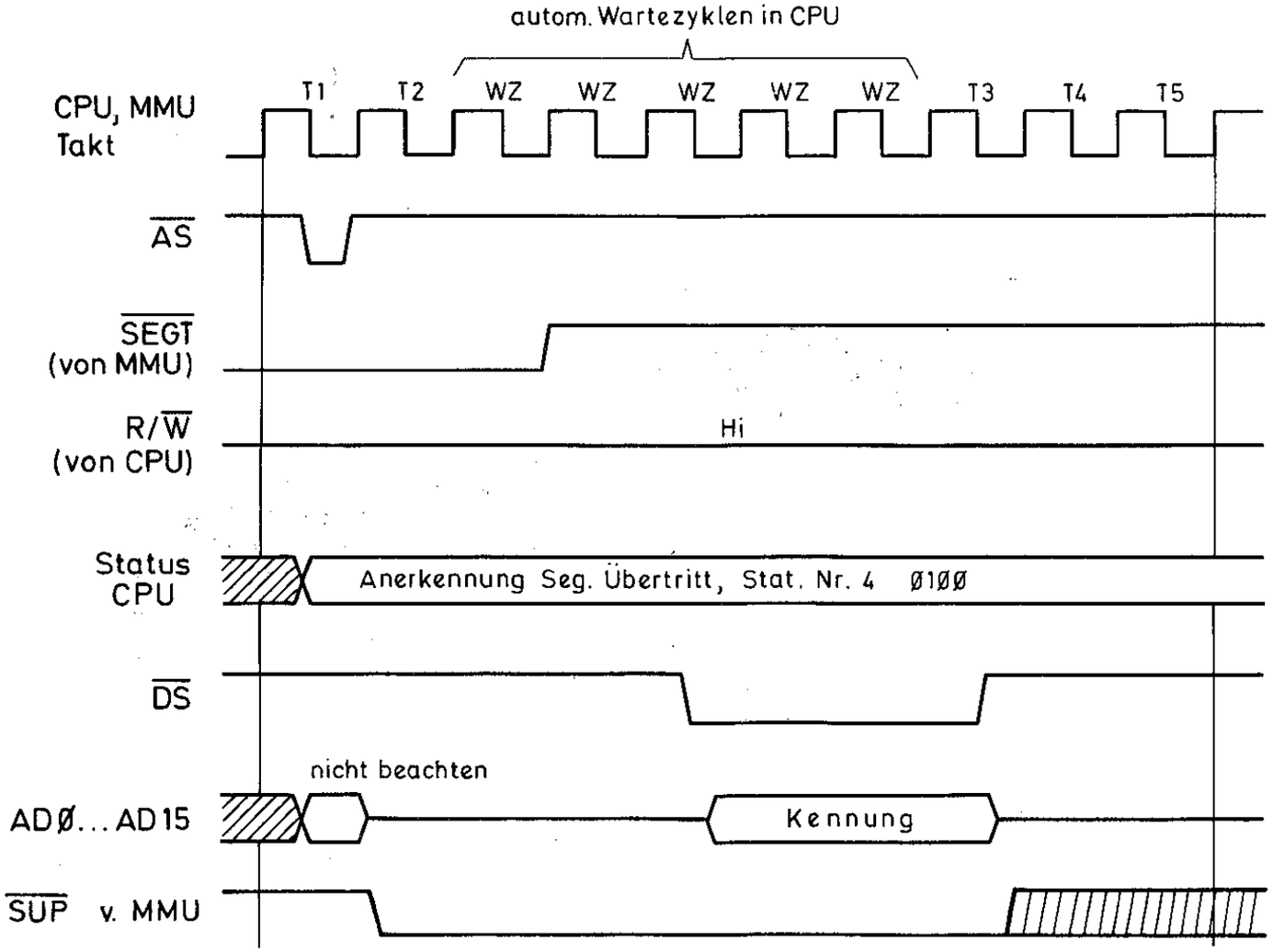


Bild 5.21

vorgestellt wurde, um ihre Raffinessen verstehen zu können, liegt ein Gedanke nicht ganz fern, die MMU mit dem Z 8000-Chip nicht von vorneherein zu vereinen, d.h. einen Chip daraus zu machen. Der Vollständigkeit halber müßte man anfügen: eine CPU mit zwei MMUs. Hier kann es nur die eine Antwort geben: **Unmöglich.** Man sollte zwar niemals nie sagen, doch ist zum gegenwärtigen Zeitpunkt eine Integration der genannten Konfiguration wirklich unmöglich. Dieses Vorhaben übersteigt derzeit jegliche Möglichkeiten der Halbleiter-Technologien bei weitem, würde man nicht die MMUs sehr stark vereinfachen.

Ein weiterer Punkt ist das MMU-Design der gegebenen Form, das die Segmentorganisation in 256 Byte-Bünden zuläßt. Warum dies? Würde man bereits bei der Adresse A0 anfangen, die logische in die physikalische Adresse umzusetzen, wäre ein 24-bit-Addierwerk notwendig. Diese umfangreiche festverdrahtete Logik erfordert eine entsprechend lange Laufzeit zur Berechnung der Überträge. Das hätte neben dem komplizierten Design eine entsprechende Verlangsamung der Adreßberechnung zur Folge. Das allein ist nicht der einzige Grund. In einem 48poligen DIL-Gehäuse wäre diese MMU nicht mehr unterzubringen; nicht wegen der erweiterten Logik, vielmehr wegen der gestiegenen Zahl erforderlicher Anschlüsse. Ein 64er Gehäuse müßte her, das in der Produktion bekanntlich sehr teuer ist. Es lohnt sich nicht weiter, darüber zu diskutieren.

Was das 48polige Gehäuse betrifft, wäre das Thema in diesem Kapitel wieder getroffen. **Bild 5.22** zeigt die Anschlüsse der MMU und ihre Bedeutung. **Bild 5.23** zeigt die Anschlußbelegung des Bausteins. Mit ihren 48 Anschlüssen zählt die MMU ohnehin nicht zu den "kleinen". Die Anschlüsse der MMU sind leicht verständlich. AD8 . . . AD15 wird direkt von der CPU herangeführt; ebenso die Segmentadreßleitungen SN0 . . . SN6, die Statusleitungen ST0 . . . ST3 und die Steuersignale $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ und N/$\overline{S}$. Der MMU-Ausgang $\overline{SEGT}$ ist ein offener Drain-Ausgang. Er wird direkt mit dem gleichnamigen CPU-Eingang verbunden. Für $\overline{SEGT}$ ist es notwendig, die Leitung mit einem 4,7 K-Widerstand auf Hi zu ziehen. Bei mehreren MMUs im System sind diese Trap-Ausgänge zu verbinden und an die CPU anzulegen (Leitung über Widerstand auf plus legen). Der Eingang DMASYNC wird von einem DMA-Baustein (Z 8016) versorgt. Ein DMA-Zugriff auf den Speicher wird von DMA in Normal- oder Systemmodus ausgeführt und meldet sich mit Lo an DMASYNC an. Eine Übersetzung der logischen DMA-Adresse erfolgt **nach** der Lo↑Hi-Flanke an DMASYNC bei der darauffolgenden Lo↑Hi-Taktflanke. Die CPU erfordert immer Hi an DMASYNC, wenn sie das Busrecht besitzt.

Der Anschluß Substrat-Auskopplung gibt die Verbindung auf den Vorspannungsgenerator. Dieser Anschluß muß ohne Belastung bleiben. Die negative Substrat-Vorspannung darf nur mit einer Kapazität von 100 nF gegen Null beschaltet werden.

Die MMU besitzt zwei Möglichkeiten eines $\overline{RESET}$. Wird am $\overline{RESET}$-Eingang ein Lo angelegt, werden die AD-Leitungen und die Adreßausgänge hochohmig. Das Betriebsart-Register wird generell auf Null gesetzt, woraus sich der hochohmige Zustand durch das MSEN-Flag erklärt. Ferner wird von den Steuerregistern das DSC-Feld auf Null gesetzt und das 8-bit-Segmentadreß-Definitionsregister (Bild 5.8) ebenfalls. Das MSEN-Flag bleibt inaktiv. Erst ein Lo an $\overline{CS}$ aktiviert das MSEN-Flag und damit die Adreßein- und -ausgänge. Ein Hardware-RESET ist folglich nur bedingt sinnvoll. Eine Initialisierung der Betriebsart muß auf $\overline{RESET}$ folgen. Ein Software-RESET mit dem Befehl 11H setzt nur das Segmentadreß-Definitionsregister auf Null. Die Funktion der MMU bleibt in diesem Fall erhalten. Die Anwendung des Hardware-RESET kann beispielsweise kombiniert mit der MMU-Freigabe erfolgen (**Bild 5.24**). Setzt man $\overline{RESET}$ zugleich mit $\overline{CS}$ auf Lo, bleibt das MSEN-Flag auf 1

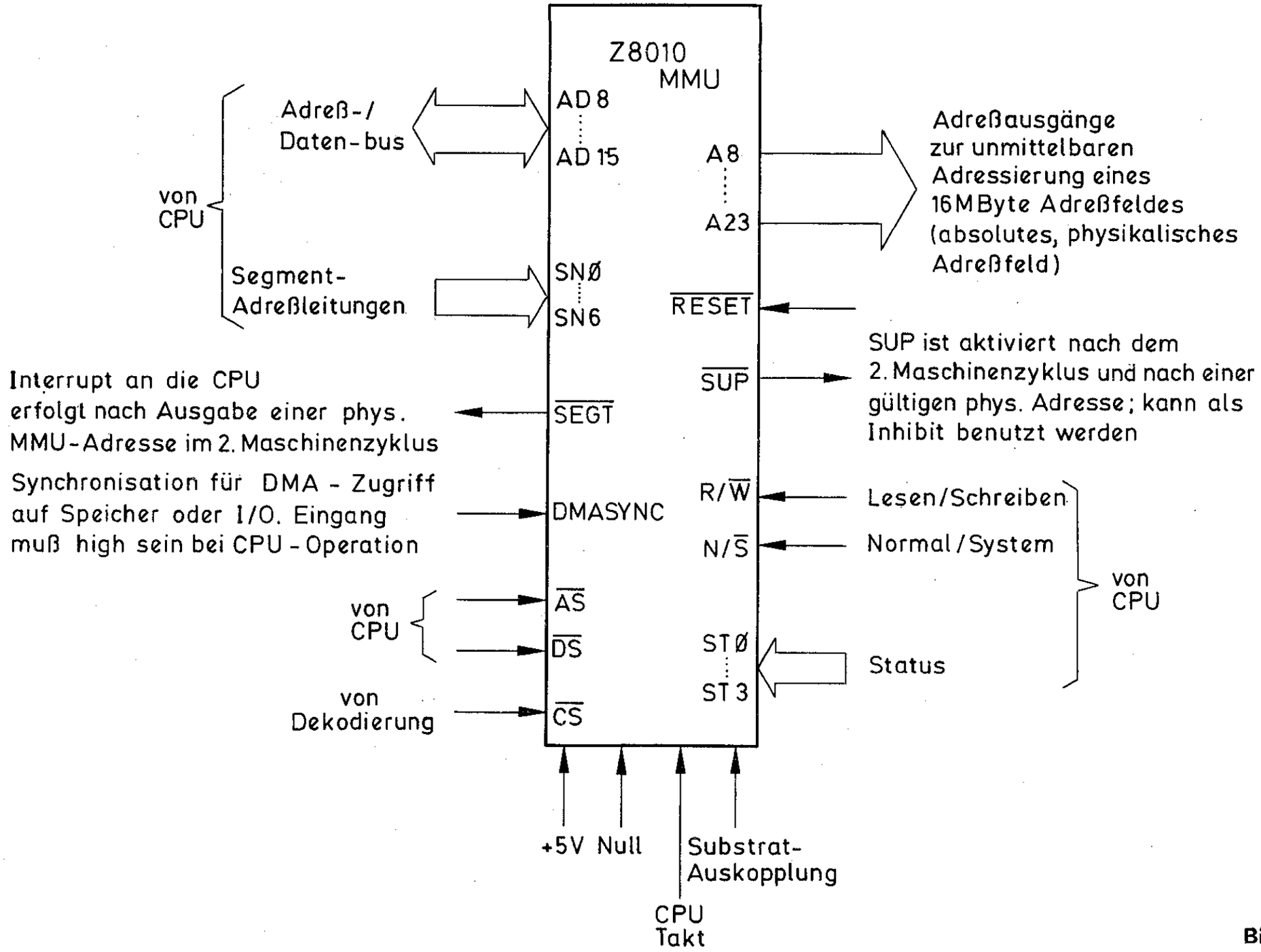

Z8010
MMU
AD 8
AD 15
Adreß-/
Daten-bus
von
CPU
Segment-
Adreßleitungen
SNØ
SN6
A8
A23
Adreßausgänge
zur unmittelbaren
Adressierung eines
16 MByte Adreßfeldes
(absolutes, physikalisches
Adreßfeld)
RESET
SUP ist aktiviert nach dem
2. Maschinenzyklus und nach einer
gültigen phys. Adresse; kann als
Inhibit benutzt werden
SUP
Interrupt an die CPU
erfolgt nach Ausgabe einer phys.
MMU-Adresse im 2. Maschinenzyklus
SEGT
Synchronisation für DMA - Zugriff
auf Speicher oder I/O. Eingang
muß high sein bei CPU - Operation
DMASYNC
R/W
Lesen/Schreiben
N/S
Normal/System
von
CPU
von
CPU
AS
DS
STØ
ST 3
Status
von
Dekodierung
CS
+5V
Null
Substrat-
Auskopplung
CPU
Takt

Bild 5.22

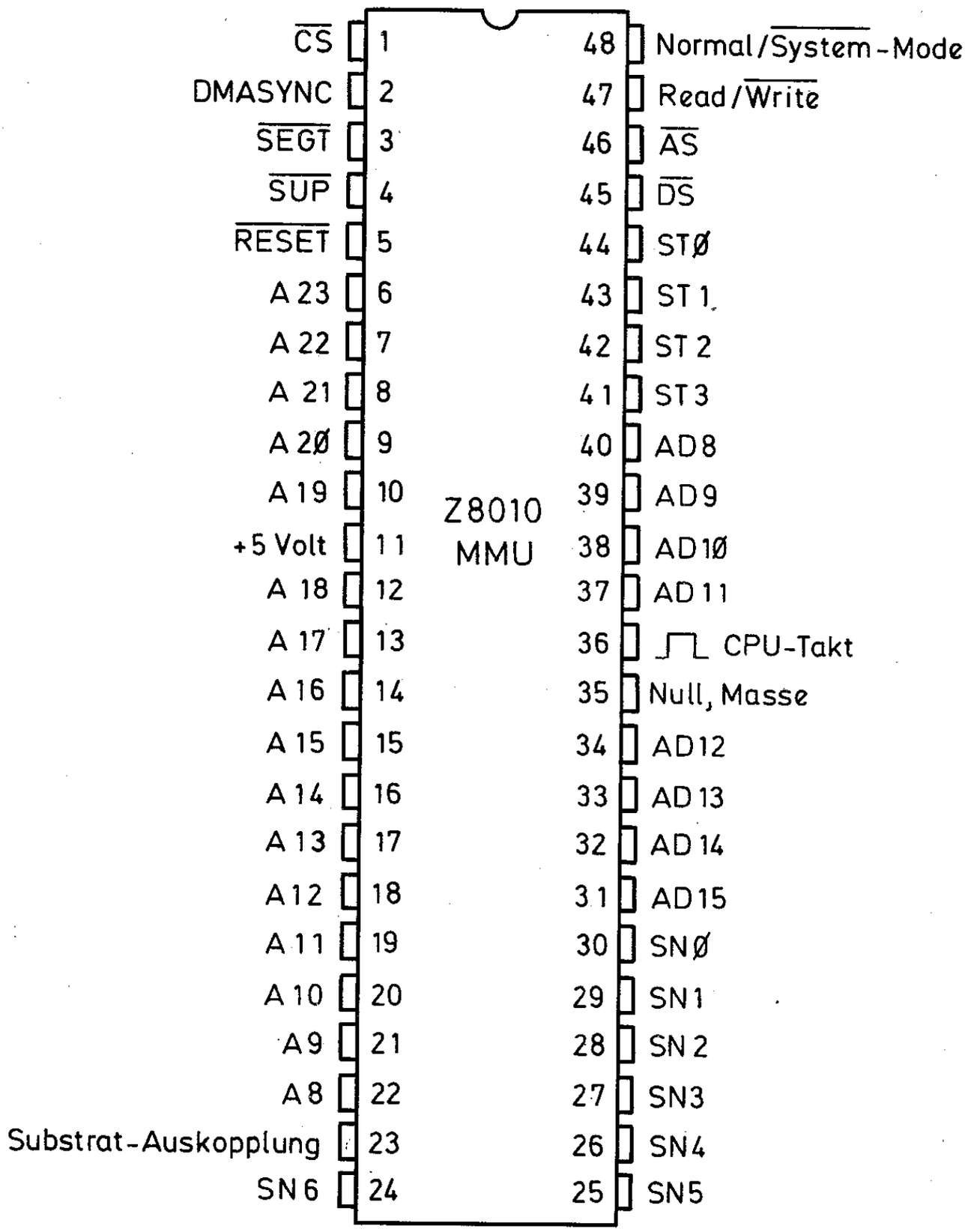

Z8010
MMU
1 CS
2 DMASYNC
3 SEGT
4 SUP
5 RESET
6 A 23
7 A 22
8 A 21
9 A 20
10 A 19
11 +5 Volt
12 A 18
13 A 17
14 A 16
15 A 15
16 A 14
17 A 13
18 A 12
19 A 11
20 A 10
21 A 9
22 A 8
23 Substrat-Auskopplung
24 SN 6
48 Normal/System-Mode
47 Read/Write
46 AS
45 DS
44 ST0
43 ST 1
42 ST 2
41 ST 3
40 AD 8
39 AD 9
38 AD 10
37 AD 11
36 CPU-Takt
35 Null, Masse
34 AD 12
33 AD 13
32 AD 14
31 AD 15
30 SN0
29 SN 1
28 SN 2
27 SN 3
26 SN 4
25 SN 5

Bild 5.23

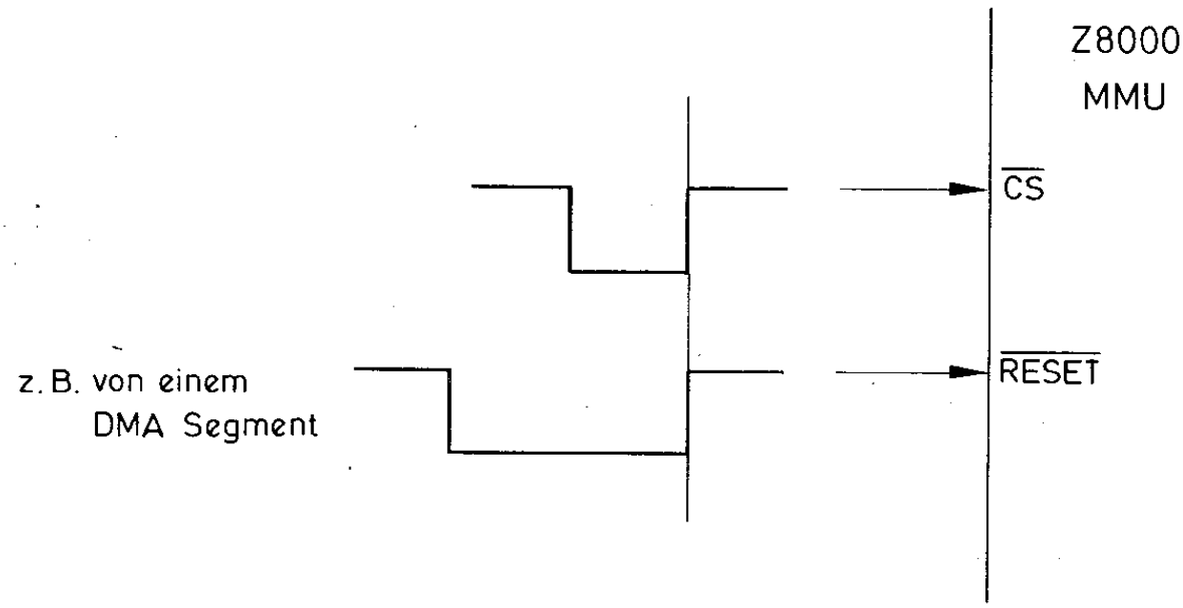


Bild 5.24

und die MMU funktionsfähig, wenn auch die vom RESET betroffenen Register nicht definiert sind. Wichtig ist dies z.B.. wenn im System nur eine MMU den Speicher verwaltet. Ist MSEN inaktiv, ist für die CPU kein Befehlsholzyklus mehr möglich, steht kein separater Speicher zur Verfügung.

5.2.1 Logik für Schreibschutz des physikalischen Speicherfeldes

Die Aktivierung des Suppress-Signals ist eine technische Reaktion der MMU auf die Verletzung eines oder mehrerer Flags im Eigenschaftsfeld eines Beschreibungsregisters. Das Auftreten von Suppress in der Mitte des zweiten Maschinezyklusses (ein klein wenig früher als der Segment-Trap) kommt zeitlich zum richtigen Zeitpunkt, um einen unzulässigen Schreibzyklus der CPU noch abzuwenden. SUP wird logischerweise nicht bei jedem Bedingungsflag aktiviert. Bei einem Nur-Lesesegment wird SUP ausgegeben, weil hier das Schreiben in das Segment verhindert werden muß. Mit SUP hat man eine Hardwareaufgabe zu lösen, die das Schreiben in ein mit RAMs bestückten Segment unterbindet. **Bild 5.25** zeigt eine Lösung. Ist der Arbeitsspeicher mit dynamischen RAMs vorgegeben, setzt man die Schreibsperre auf das Signal zur Spaltenfreigabe an (CAS). Liegt an dem ODER-Gatter ein Lo für R/W und wurde auf Grund einer Verletzung der Segmentbedingung SUP gleich Lo, wird an dem oberen NAND-Gatter kein Lo für eine Spaltenfreigabe durchgeschleust. Der Schreibzyklus ist damit unwirksam geworden. Die Schaltung läßt sich sehr leicht noch in das Speicherwerk nach Bild 4.42 einfügen. Der Speicherzugriff wird gerade um 12 ns beeinflußt.

Das Suppress-Signal wird bei einem Segmentzugriff aktiviert, wenn

- das RD-Flag gesetzt war;
- das CPUI-Flag gesetzt war (bei DMA-Zugriff);
- das DMAI-Flag gesetzt war (bei CPU-Zugriff);
- das EXC-Flag gesetzt war.

Das Suppress-Signal wird auch aktiviert,

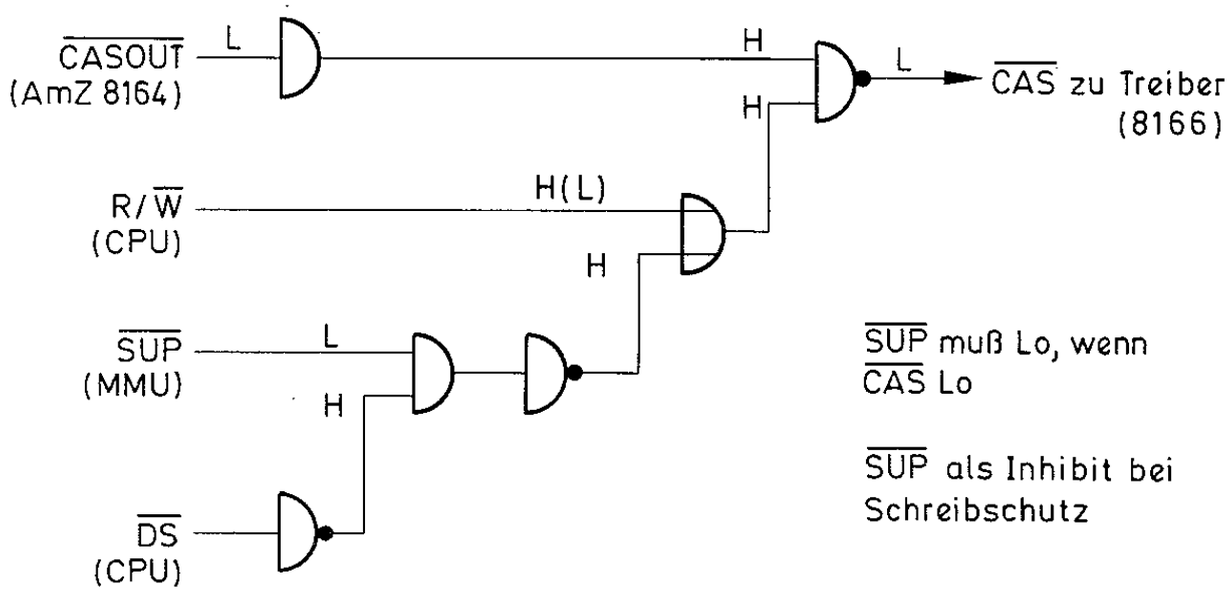


Bild 5.25

wenn im Segmentadressen-Definitionsregister (Bild 5.8) das FATL-Flag gesetzt wurde. Dies wird, wie schon genannt, dann gesetzt, wenn während einer Segmenttrap-Serviceroutine ein unzulässiger Schreibzyklus von der CPU ausgelöst wurde.

$\overline{SUP}$ wird nicht aktiviert, wenn das DIRW-Flag verletzt wird. Das geschieht deshalb nicht, damit bei "Stack-push"-Operationen keine Störung beim Laden des Stapelsegments auftritt. Bei "Stack-push"-Operationen wird das betreffende Register oder die Stapelzeiger (Normal, System) bekanntlich dekrementiert, bevor der Stapel geladen wird. Deshalb muß ein Stapelsegment immer nach kleineren Adreßwerten hin erweitert, bzw. vergrößert werden.

Im folgenden Abschnitt wird die Grundkonfiguration von CPU und MMU betrachtet.

5.3 ANSCHLUSS-SCHEMATIK DER Z 8010-MMU MIT DER Z 8001-CPU

Die Notwendigkeit einer Speicherverwaltung wurde bereits dargestellt. Die Vorteile einer Segmentierung des Speicherwerks bei Mehrbenutzeranlagen und der daraus erwachsende Zugriffsschutz wurde hinreichend betrachtet. Die Umsetzung dieser technischen Notwendigkeit zeigt sich bei einer Reihe von Minicomputern in einem beträchtlichen Hard- und Softwareaufwand. Der Grund, für die Umsetzung die Z 8010-MMU einzusetzen, ist rein ökonomisch. Die MMU erfüllt alle gestellten Anforderungen und bietet die Realisierung in einfachster Art. Das gilt für die Hardware und die Software gleichermaßen. Die Hardware betreffend werden zwei Varianten vorgestellt.

5.3.1 Ein MMU-Baustein im Verbund mit der Z 8001-CPU

Bild 5.26 zeigt die einfache Konfiguration. Die entsprechenden CPU-Steuerleitungen sind direkt an die MMU anzulegen. Will man auf einen Hardware-RESET der MMU verzichten, legt man den Anschluß statisch auf Hi. Für den Eingang DMASYNC gilt das ebenso, wenn sich kein DMA-Baustein im System befindet. Die MMU-Freigabe bei der Initialisierung wird an $\overline{CS}$ mit der invertierten Adreßleitung A1 bewirkt. Die Segmenttrap-Leitung $\overline{SEGT}$ von der MMU an die CPU ist mit einem Widerstand auf Hi zu setzen. Die Taktversorgung erfolgt phasengleich an CPU und MMU gemeinsam. Wie aus der Schaltung weiter zu entnehmen ist, wird der AD-Bus mit Hilfe der Übertrager (AmZ 8104) auf Daten in der vollen Breite separiert und mit einem Zwischenspeicher-Baustein bezüglich der Adressen A0 . . . A7 separiert. Die Hardware-Konfiguration zeigt nicht das Initialisierungs-PROM für CPU und MMU. Die Initialisierung kann über die MMU unmittelbar nach dem Einschalten des Systems erfolgen, wenn im Betriebsart-Register das TRNS-Flag auf Null und das MSEN-Flag auf Eins gesetzt ist. Dies ist jedoch nach unmittelbarem Einschalten nicht gewährleistet. Sicher ist es nur, wenn man die beiden Flags definiert setzt. Bleibt als Konsequenz, daß es empfehlenswert ist, das Initialisierungsprogramm im logischen Adreßfeld anzulegen.

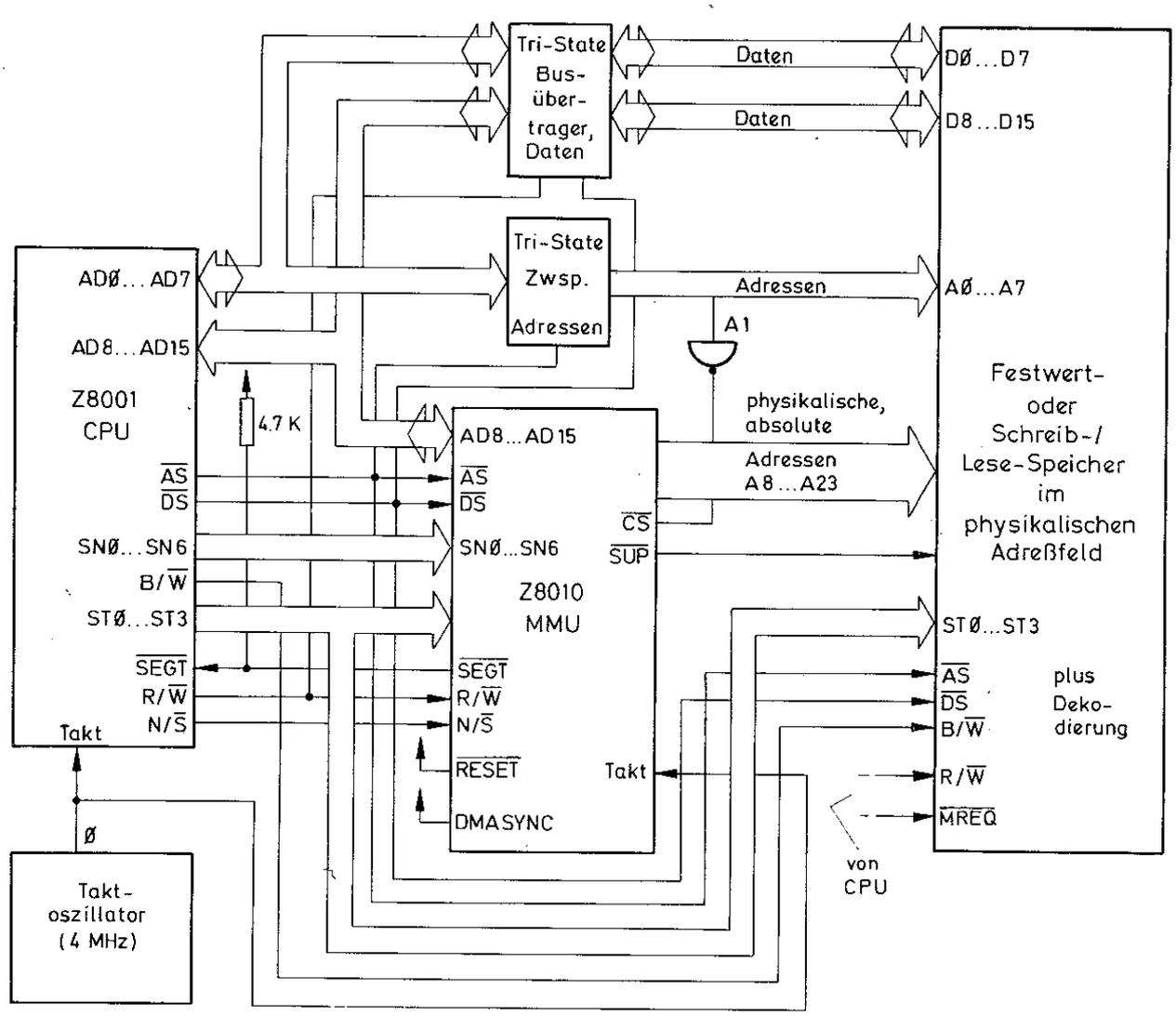


Bild 5.26

5.3.2 Zwei MMU-Bausteine im Verbund mit der CPU

In **Bild 5.27** schließlich wird eine CPU/MMU-Hardwarekonfiguration vorgestellt, die die Verwaltung aller 128 CPU-Segmente gestattet. Das System wird ferner von dem DMA-Steuerbaustein Z 8016 unterstützt, der, wie die CPU, einen logischen Adreßraum von 8 MB adressieren kann und für einen Betrieb mit den MMUs zugeschnitten ist. Das Suppress-Signal kann direkt an den DMATC angelegt werden. Die beiden $\overline{\text{SEGT}}$-Ausgänge der MMUs werden über Widerstände auf Hi gezogen und an die CPU angelegt. Das UND-Gatter kann auch entfallen. Für die Tandemanordnung der MMUs gilt, daß die Treiberleistung der CPU ausreichend ist, sowohl die MMUs und den DMATC direkt anzuschließen. Das gilt für den AD-Bus, die Segmentleitungen und die Steuersignale. Würden vier MMUs und mehr angeschlossen (plus DMA), wird bei weiteren zusätzlichen Belastungen der Busleitungen das Treiben nötig werden. Ansonsten gelten die bereits für die vorangegangene Version gemachten Anmerkungen. Das Zusammenlegen der einzelnen Adreßausgänge beider MMUs ist erlaubt, denn bei der Segmentaufteilung in die untere und obere Hälfte (URS-Flag Null bzw. Eins) sind die Ausgänge des jeweilig inaktiven Bausteins hochohmig.

Eine Schlußbemerkung sei zu diesem Kapitel und der Schaltung in Bild 5.27 noch angefügt. Mit dieser Schaltung ist eine Computer-Hardware absolut kapitaler Prägung angedeutet. Ein Z 8001-System dieser Art kann in seiner Leistungsfähigkeit nur noch mit Großrechnern verglichen werden. Ein Z 8001-System dieser Art wird das Maß an Computertechnik der 80er Jahre repräsentieren, an dem entsprechende Maschinen dieser hohen Klasse gemessen werden müssen.

Auf kleinstem Raum ist eine Zentraleinheit unterzubringen, die für die späten 70er Jahre noch seines gleichen sucht. Vier 48polige DIL-Keramikbausteine bilden den Kern eines gewaltigen Computers. Hiermit sind die Zeichen für die Zukunft gesetzt. Bleibt hierfür noch die Entwicklung der Software. Betriebssysteme für Computer solchen Kalibers orientieren sich nicht mehr mit Vorzug auf die Hardware (im Gegensatz zu früheren Hardwareentwicklungen), sondern mit großer Priorität spezifisch darauf, welche Bedienungswünsche ein Anwender an solch ein System stellt. Die außerordentlichen Raffinessen der Bausteine dieses Computers gestatten es, das Betriebssystem, trotz eines gut überschaubaren Aufwands, auf einen Bedienungskomfort hinzutrimmen, daß auch ein "blutiger" Laie mit solch einem Computer sehr effektiv arbeiten kann. Eine Hardware dieser extrem hohen Klasse erlaubt es, die Bedienung dem Anwender "quasi auf den Leib zu schneidern". Dies sind die Erwartungen, die man stellen darf.

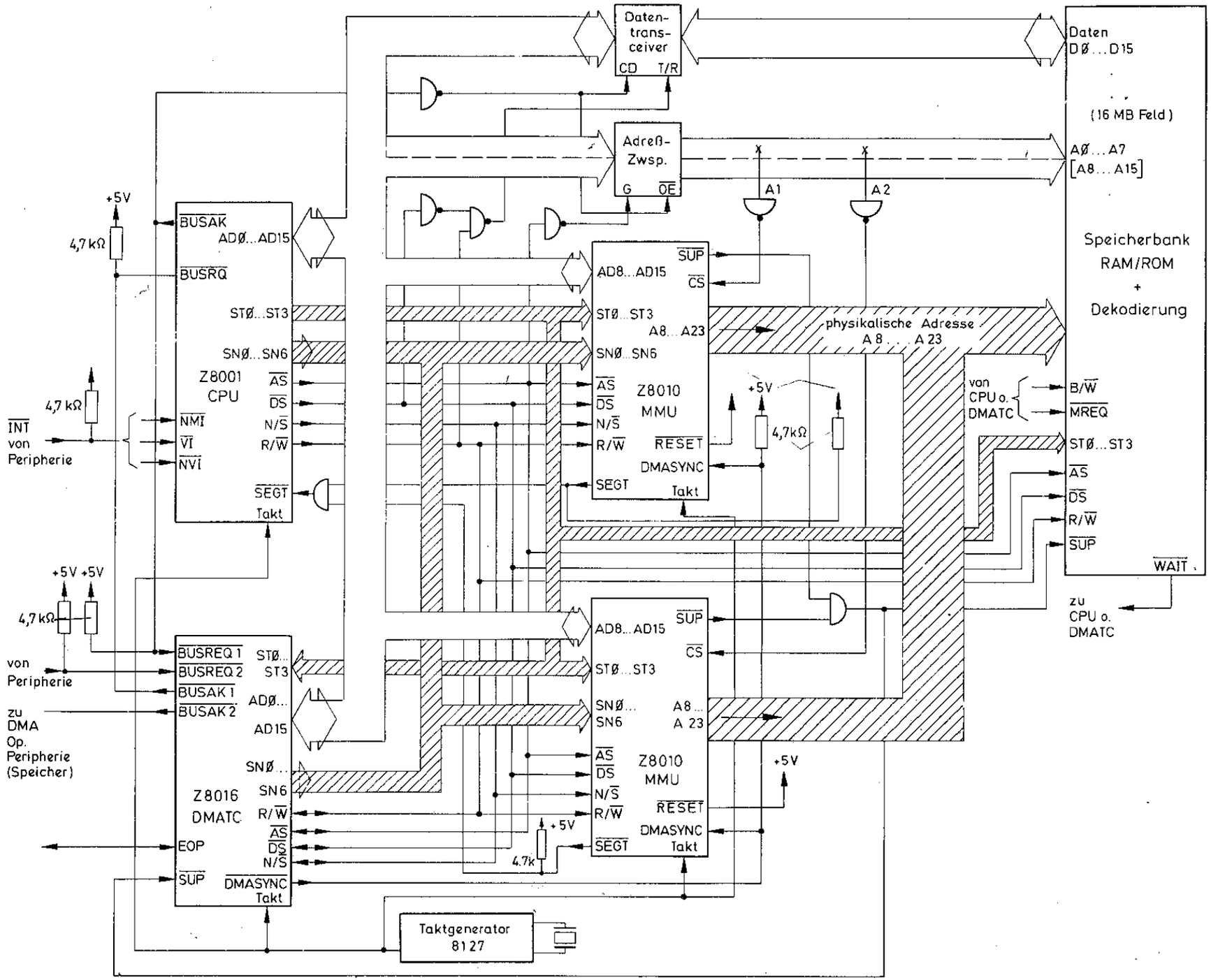

Daten-
trans-
ceiver
CD
T/R
Daten
DØ...D15
(16 MB Feld)
Adreß-
Zwsp.
G
OE
AØ...A7
[A8...A15]
A1
A2
Speicherbank
RAM/ROM
+
Dekodierung
+5V
4,7 kΩ
BUSAK
ADØ...AD15
BUSRQ
STØ...ST3
SNØ...SN6
Z8001
CPU
AS
DS
N/S
R/W
NMI
VI
NVI
SEGT
Takt
INT
von
Peripherie
AD8...AD15
SUP
CS
A8...A23
Z8010
MMU
RESET
DMASYNC
physikalische Adresse
A 8 ... A 23
von
CPU o.
DMATC
B/W
MREQ
WAIT
zu
CPU o.
DMATC
+5V +5V
BUSREQ 1
BUSREQ 2
BUSAK 1
BUSAK 2
Z8016
DMATC
EOP
SUP
AD15
SN6
4.7k
zu
DMA
Op.
Peripherie
(Speicher)
Taktgenerator
8127

Bild 5.27

Assembler-Programmiertechnik mit den Z8000-Befehlen

6

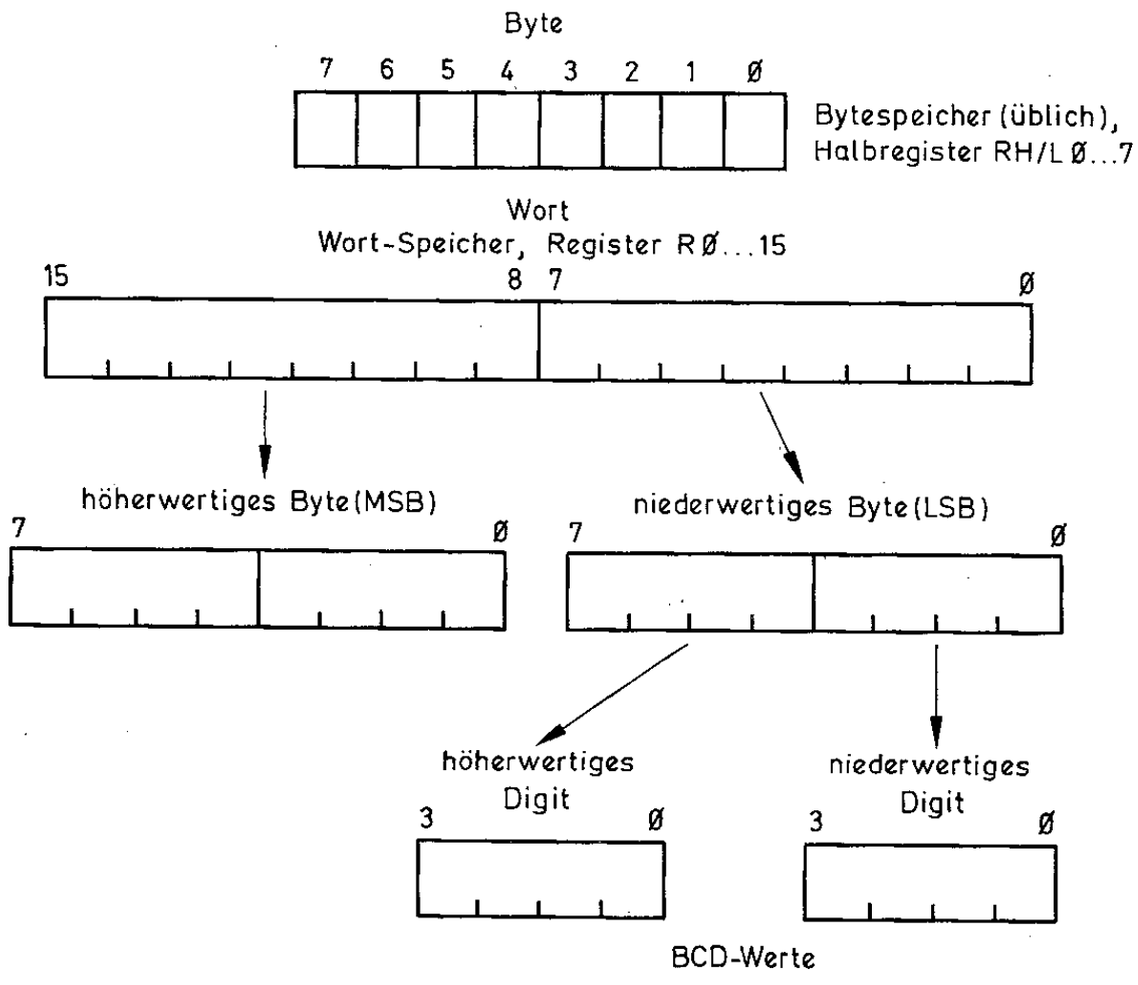
"Ohne Fleiß kein Preis und Übung macht den Meister". Diese beiden Sprichwörter haben für den Umgang mit der Assemblersprache ihre berechtigte Gültigkeit. Die Assembler-Sprache ist für die Z 8000-CPU nicht schwieriger zu erlernen als für den 8080 oder den Z 80. Der Z 8000 hat wohl einen größeren Befehlssatz als die kleinen; man hat jedoch im Gegensatz dazu weit weniger spezifische Bedingungen zu beachten, die die Programmierung bei den kleinen tatsächlich erschweren. Letztlich ist immer wieder entscheidend, welche Programmieraufgaben man lösen muß. Folglich sind aus den neun Befehlsgruppen für den einen oder anderen Fall nur ein Teil der Befehle brauchbar. Auf diese Befehle kann man sich im einzelnen konzentrieren. Der Aufwand an neu zu lernendem Stoff steht in Bezug zu der dafür gebotenen Leistung der CPU in einem guten Verhältnis.

6.1 ELEMENTARE OPERATIONEN (LADEN, RECHNEN, SCHIEBEN, TRANSFERIEREN, SEGMENTSPRUNG)

Die CPU manipuliert Operanden (einzelne Werte) nach den Wünschen des Anwenders. Diese allgemein unter dem Sammelbegriff Daten zusammengefaßten Operanden müssen nicht das vorgegebene Datenformat von 16 bit haben. Der Z 8000 bearbeitet fünf verschiedene Datentypen, womit die Länge eines Operanden gemeint ist. Dies sind

- einzelne Bits (in einem Wort)
- 4 bit Wörter, speziell in BCD codierter Art
- 8 bit Wörter, generell als Bytes bezeichnet
- 16 bit Wörter, als Standardformat zu bezeichnen
- 32 bit Wörter, als längstes Format eines Operanden

Bild 6.1 zeigt diese möglichen, unterschiedlichen Längen von Operanden. Die Bearbeitung der unterschiedlich langen Operanden wird in den passenden Registern der CPU vorgenommen. Für Bit-, BCD- und Byte-Operanden sind die Register RL0 . . . 7 bzw. RH0 . . . 7 verwendbar; für Wortoperanden ist R0 . . . R13/14 verwendbar; für Langwort-Operanden die Doppelregister RR0, 2, 4, 6, 8 und 12. Alle Bewegungen von Operanden werden von Register zu Register, von Speicherplatz zu Register und von Speicherplatz zu Register zu Speicherplatz ausgeführt. Bytes ergeben den Wertbereich von 0 . . . 255 (bzw. −128 . . . +127 mit Vorzeichen). Für 16 bit Worte wird ein Bereich von 0 . . . 65535 (bzw. −32768 . . . +32767 mit Vorzeichen) möglich. Für 32 bit Worte wird schließlich ein Bereich von 0 . . . 4 294 967 295 (bzw. −2 147 483 648 . . . +2 147 438 647) möglich.


Byte
7 6 5 4 3 2 1 Ø
Bytespeicher (üblich),
Halbregister RH/L Ø...7
Wort
Wort-Speicher, Register RØ...15
15
8 7
Ø
höherwertiges Byte(MSB)
niederwertiges Byte(LSB)
7
Ø
7
Ø
höherwertiges
Digit
niederwertiges
Digit
3
Ø
3
Ø
BCD-Werte

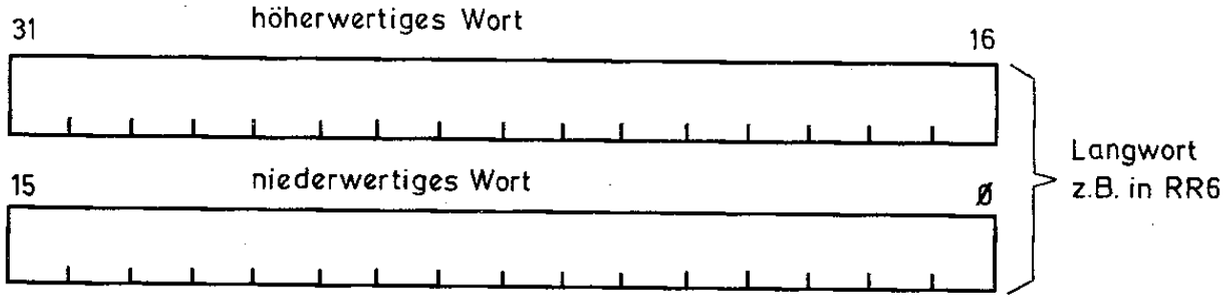
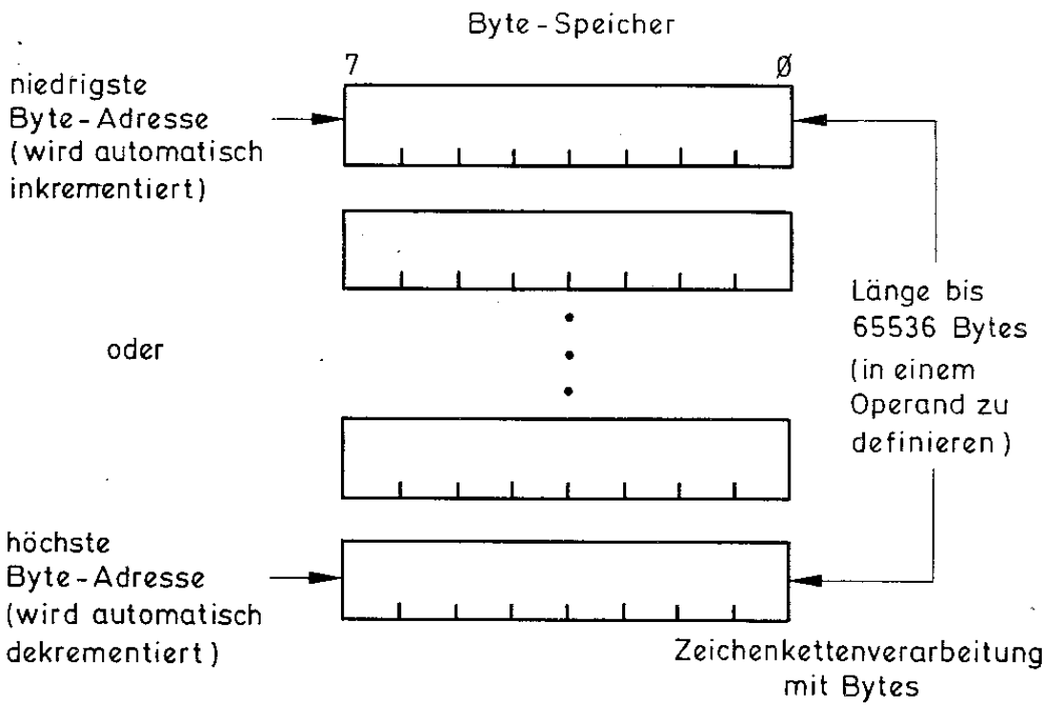
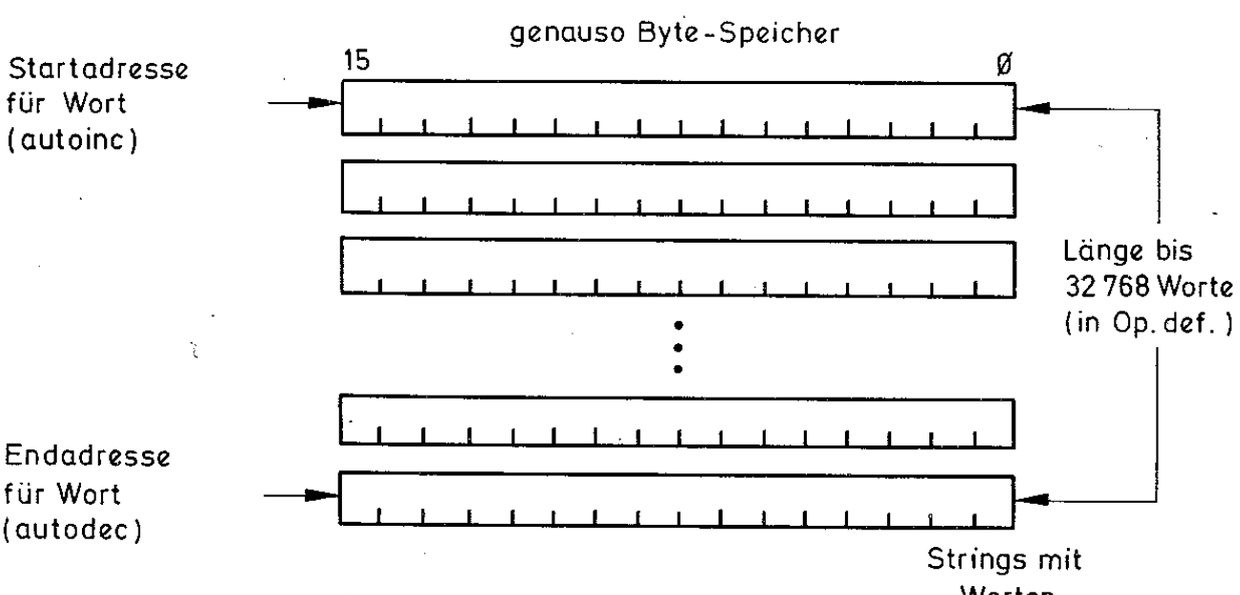

31
höherwertiges Wort
16
15
niederwertiges Wort
Ø
Langwort
z.B. in RR6

Bild 6.1





Befehle: CPIR (Wort); CPIRB (Byte)
Zeichenkettenvergleich, wiederholend
mit Adreßerhöhung um 1 (Byte), um 2 (Wort)

CPSDR (Wort); CPSDRB (Byte)
... Adreßerniedrigung um 1 (Byte), um 2 (Wort)

Bild 6.2

Damit ist der Z 8000 noch längst nicht am Ende. Wie aus Kapitel 3 zu entnehmen ist, kann die CPU sehr komplizierte Zeichenkettenbefehle bearbeiten. In **Bild 6.2** ist angedeutet, daß beispielsweise der Vergleich von Zeichenketten byteweise und wortweise möglich ist. Zeichenketten (Strings) werden verständlicherweise nur im Speicher abgelegt. Als reine String-Operationen verstehen sich die Vergleichsbefehle und die "Translate"- bzw. "Translate and Test"-Befehle. Beide Arten sind Doppelwort-Befehle. Bezüglich der Zeigeradressen auf die Strings sind diese entweder automatisch inkrementierend (um eine bzw. zwei Stellen erhöhend) oder dekrementierend ausführbar. Die Adressierungsart beider Befehlstypen ist indirekt. Es müssen folglich Registerplätze für die String-Zeigeradressen reserviert werden. Die "Translate and Test"-String-Befehle sind nur mit Bytes möglich.

Die Bearbeitung einzelner Bits erfolgt durch die Befehle "Test Bit (statisch, dynamisch), Reset Bit (statisch, dynamisch, Set Bit (statisch, dynamisch) und Test and Set".

Eine wichtige Bemerkung sei für alle Beispiele zu beachten, die nach dem Z 8000-Assembler gegeben werden. Bei den bisherigen Beispielen konnte man dies bereits bemerken, ohne daß darauf besonders aufmerksam gemacht wurde. Die Reihenfolge in der Assemblerschreibweise ist grundsätzlich

Opcode Ziel , Quelle .

Für den Opcode wird die Befehlskurzbezeichnung eingesetzt (ADD, SUBL, CALR, SRLB . . .); für das Ziel steht eine Registerangabe oder eine Adresse und für die Quelle nach dem Komma steht ebenfalls eine Registerangabe, eine Adresse oder ein Datenwert.

Beispiel:

```
ADDL    RR 0 ,#%A1FF3BCF
```

Dies gilt beim Assembler immer. Programmiert man im HEX-Code, merkt man davon nichts, wenn es in der Befehlsdefinition auch einmal umgekehrt

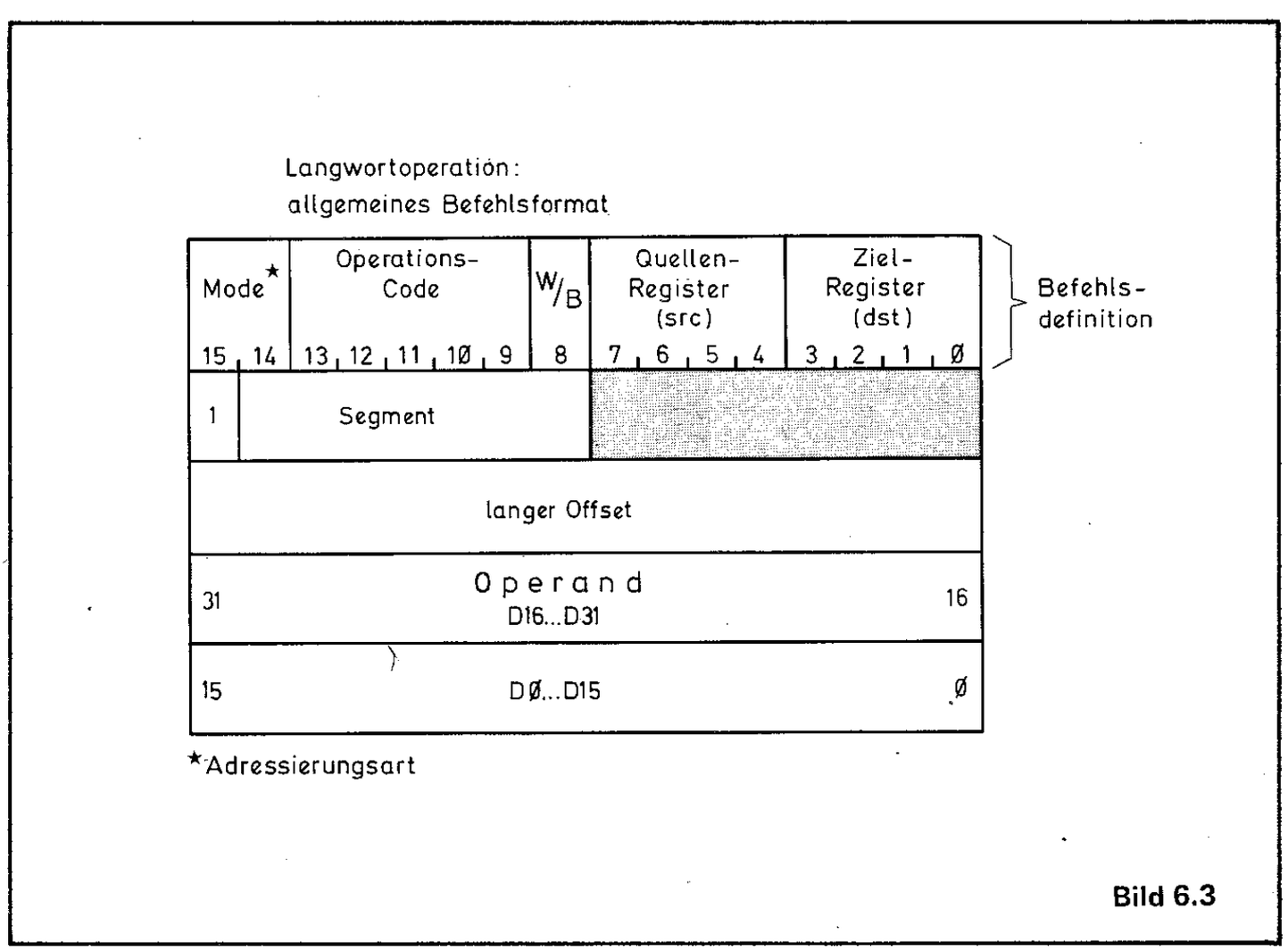
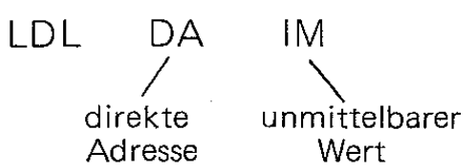


Bild 6.3

lautet. Beim Additionsbefehl steht der Quellabschnitt zuerst in der Befehlsdefinition. Bei LDK (lade Konstante) ist es umgekehrt.

Laden:
Die sicherlich für jedes Programm benötigten Befehle sind Lade-Befehle. Sie sind in vielfältigen Variationen anzuwenden. Die Quellangabe kann in 7 Adressierungsarten definiert werden, die Zielangabe in 6 Arten. Es können Bytes, 16 bit und 32 bit Worte geladen werden. **Bild 6.3** zeigt den längsten Befehl, der beim Z 8000 möglich ist. Dieser Lade-Befehl gibt an, daß ein fester 32-bit-Wert in einen Speicherplatz eines Segments geladen werden soll. Für die Quelle gilt in diesem Fall die unmittelbare Adressierungsart; für das Ziel gilt die direkte Adressierungsart. Die Assemblerschreibweise lautet demnach folgendermaßen, wenn dies in einer allgemeinen Art angegeben werden soll:



In einem speziellen Fall kann dafür der Assembler-Syntax

LDL ≪48≫ % 1FC1, # % A1FF3BCF

angegeben werden. ≪48≫ steht für das Segment, % 1FC1 für die Offsetadresse und # % A1FF3BCF für den direkten Wert. Alle fünf Wörter dieses Befehls werden aus dem Programmcode-Speicher geholt. Zum Bild 6.3 ist anzumerken, daß immer erst die Adresse und dann der Operand einzugeben ist. Hat man einen Assembler zur Verfügung, muß man sich nur nach dem gezeigten Syntax halten.

Die unmittelbare Adressierungsart hat für Lade-Befehle eine Bedeutung bei Initialisierungsroutinen. Um das System nach dem Einschalten auf definierte Anfangsbedingungen zu setzen, wird direkt aus dem Programmspeicher ein Anfangswert geholt. Für das Ziel wird generell ein internes Allzweckregister adressiert. Der Grund liegt in der Anwendung der LDCTL-Befehle (laden des Steuerworts). Die CPU-Steuerregister sind nur von einem Allzweckregister ladbar. Ein Beispiel zeigt das Laden des Flag- und Steuerwortregisters (FCW). .

```
LD    R1, # % C000
LDCTL FCW, R1
```

Der explizite Wert C000H besagt, daß im FCW Bit 15 und 14 auf Eins gesetzt werden, d.h. segmentierter Modus (Z 8001) und System-Modus. VIs und NVIs werden nicht akzeptiert. Ferner sind alle Flags auf Null gesetzt.

Rechnen:
Rechnen mit dem Z 8000 bedeutet die Verfügung über die vier Grundrechnungsarten mit jeweils einem Befehl. Rechnen mit der CPU bedeutet auch die Rotation eines Digits (4 bit BCD-Wert) in zwei Byteregistern. Mit RLDB bzw. RRDB lassen sich BCD-Multiplikationen bzw. -Divisionen mit dem Faktor 10 ausführen. Mit dem Befehl DAB lassen sich binäre Additionen bzw. Subtraktionen in BCD-Werte umrechnen. Das ist der Umriß des Spektrums bezüglich rechnen mit der CPU. In den folgenden **Bildern 6.4, 6.5, 6.6, 6.7** und **6.8** sind einige binäre Rechenoperationen grafisch dargestellt. Die beiden Additionen erfolgen in Bild 6.4 von Register zu Register und in Bild 6.5 von einem Register und einem direkten Wert (unmittelbarer Operand) mit Ziel in dem ersten Operandenregister. Für die Multiplikation sind vergleichbare Beispiele dargestellt. Folgende Beispiele beziehen sich auf Rechenarten mit einer anderen Adressierungsart der Quelle.

```
ADD   R4  , @R12              indirekt
ADD   R4  , ≪48≫ % 1FC1       indiziert,
                              segmentiert
SUB   R4  , @R12              indirekt
SUB   R4  , % 1FC1            direkt
MULTL RQ0, % 1FC1             direkt
DIV   RR0, % 1FC1 (R10)       indiziert,
                              nicht segmentiert
DIVL  RQ0, % 1FC1             direkt
```

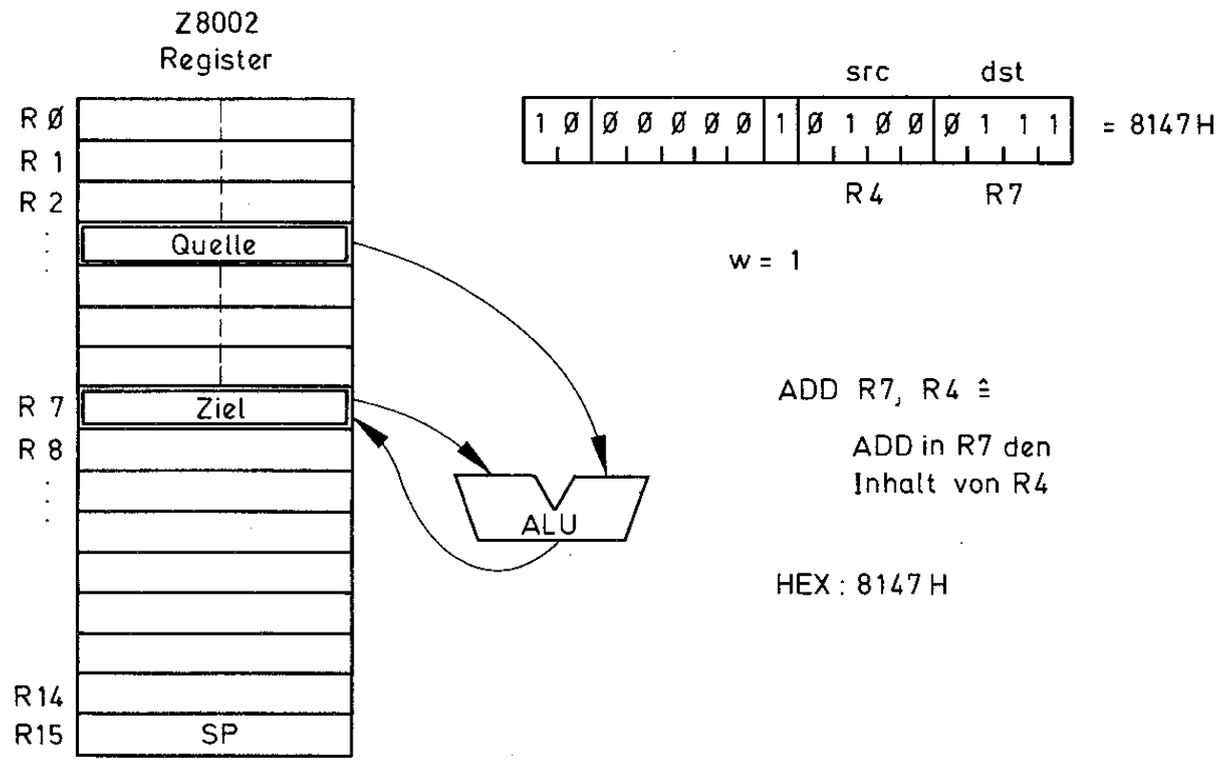


Bild 6.4

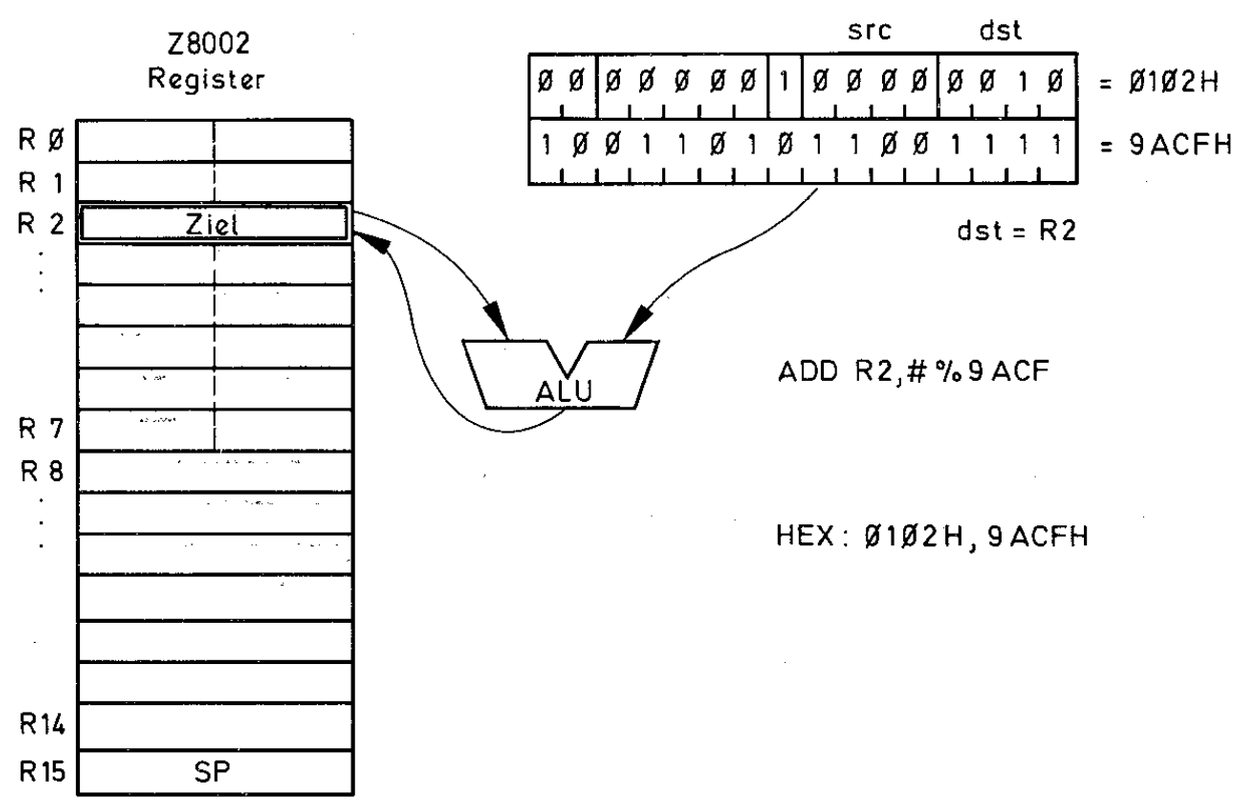


Bild 6.5

Die bereits genannten Rotationsbefehle für Digits können für eine BCD-Arithmetik benützt werden. Das folgende Beispiel wäre normalerweise in Form eines Flußdiagramms darzustellen. Die einzelnen Blöcke würden jedoch zu viel Text enthalten müssen, um verständlich genug zu sein. Deshalb wird hier etwas einfacher verfahren. Das Beispiel beschreibt eine BCD-Multiplikation mit dem Faktor 10, die auf beliebig viele Stellen auszudehnen ist.

START

beliebiges Register als Schleifenzähler benützen und mit einem Wert laden, der der Hälfte der gewünschten Zahl von Digits entspricht.

beliebiges Register als Digitzeiger benützen und auf das niederwertigste Paar aller Digits mit der entsprechenden Adresse laden

beliebiges Byteregister an allen Stellen mit Nullen laden (z.B. RH1)

* anderes Byteregister mit Inhalt der Adresse des Digitzeigers laden (z.B. RL1)

Linksverschiebung mit dem RLDB-Befehl und RH1 als Zielplatz, RL1 als Quellplatz

den Inhalt des Quellplatzes an die Adresse im Digitzeiger ablegen

Digitzeiger-Adresse um Eins erniedrigen

Schleifensteuerung mit einfachem Branch-Befehl DJNZ anwenden und ebenfalls dekrementierten Schleifenzähler auf Null überprüfen

Ist Schleifenzähler nicht Null, Programmzähler auf Adresse von * setzen

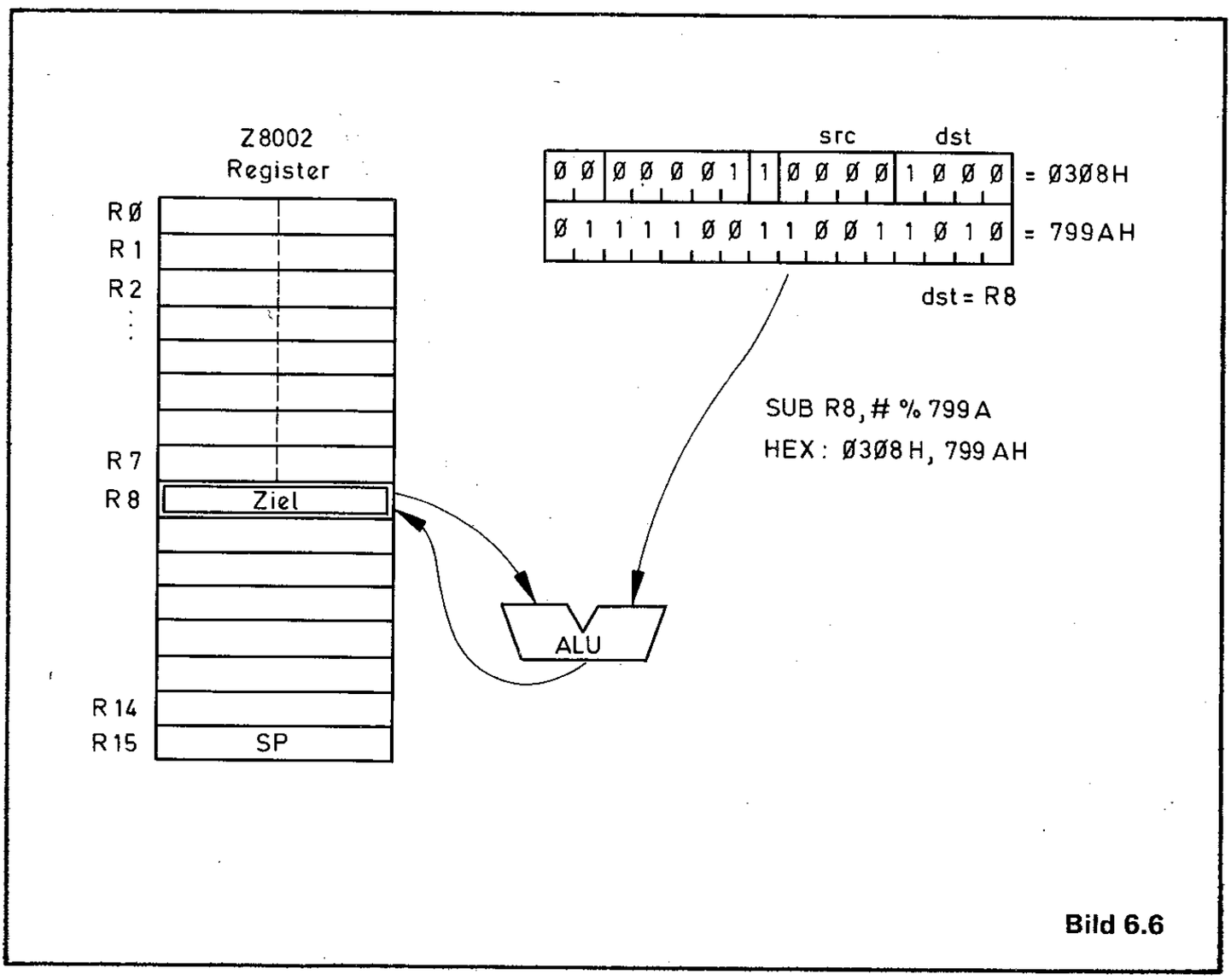


Bild 6.6

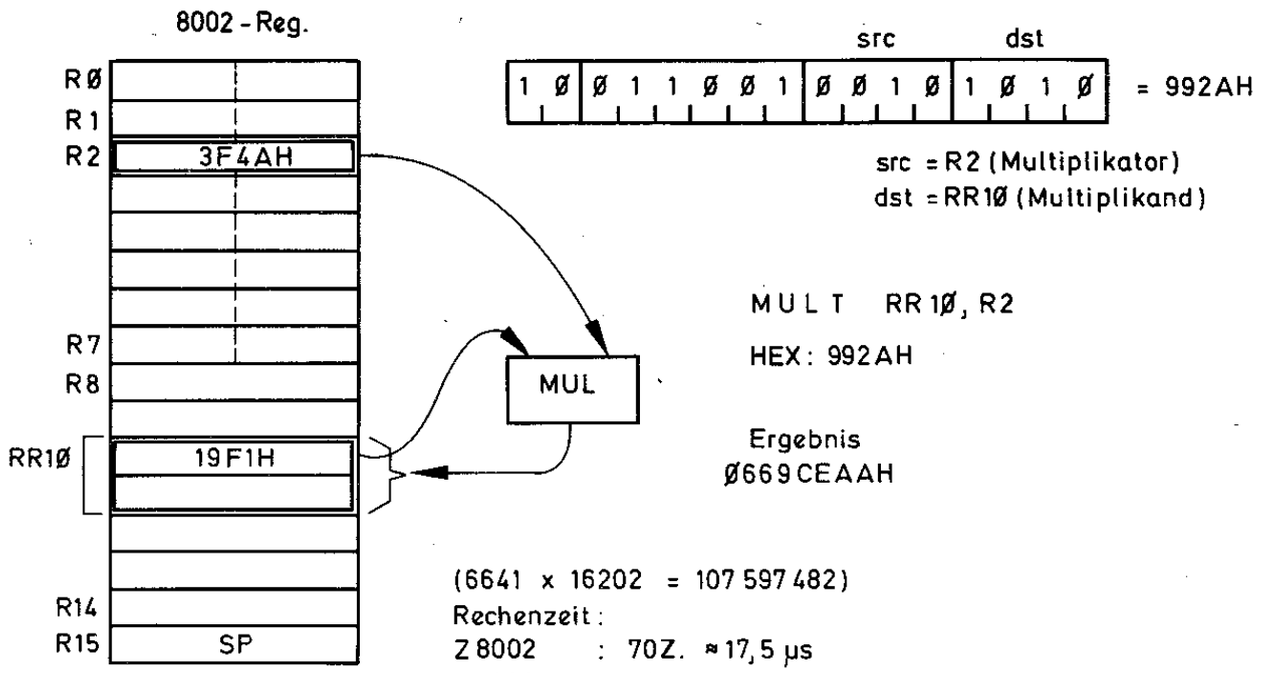


Bild 6.7

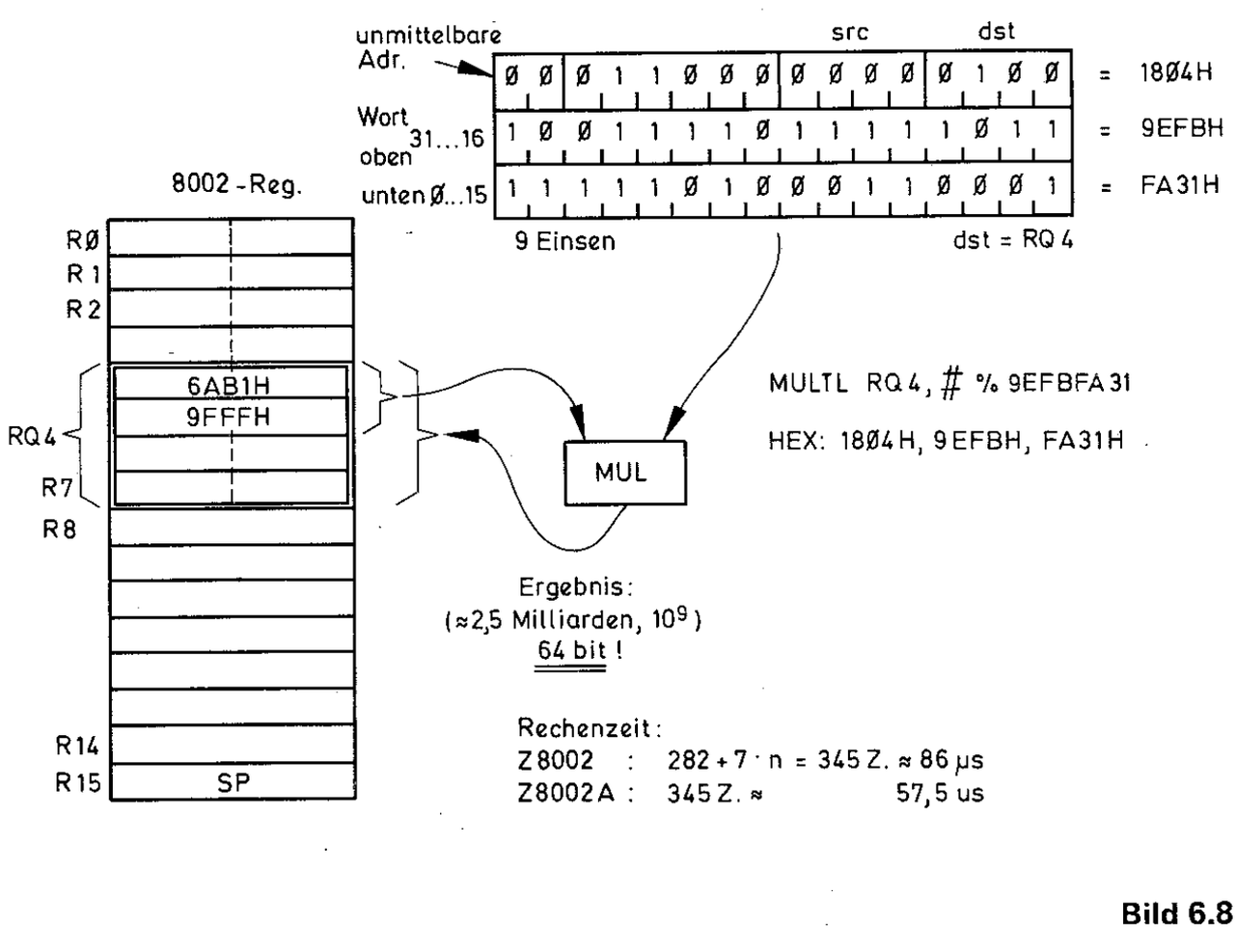


Bild 6.8

Mit diesem Ablauf ist eine digitale Multiplikation mit dem Faktor 10 eines beliebig langen BCD-Wertes auszuführen. Das Zielplatzregister enthält immer den 4-bit-Übertrag. Dieser Übertrag wird jeweils durch RLDB an die richtige Stelle des Doppeldigits plaziert. Ist der Schleifenzähler auf Null erniedrigt worden, kann das Programm noch einmal gestartet werden, womit dann der ursprüngliche BCD-Wert mit dem Faktor 100 multipliziert worden wäre. **Bild 6.9** zeigt das Verschieben eines Digits in zwei Byteregistern. Vom Zielteil (dst) wird das Digit (Bit 0 . . . 3) in den Quellteil (arc) auf die Stellen 0 . . . 3 versetzt. Die Pfeile zeigen die Versetzung der Digits an. Dieser Vorgang wird bei der Befehlsausführung einmal durchgeführt. Ablauftechnisch ist für RH und RL eine gleiche Nummer von Vorteil. Ausführungstechnisch dürfen für die Byteregister beliebige Werte in den angegebenen Grenzen eingesetzt werden. Ein Programmbeispiel zeigt die Anwendung des RLDB-Befehls:

```
                LDL  RR2,#%000B1FC1
                LDB  RH0,#00
$ SSXX →        LDB  RL0,@R3
          RLDB  RH0,RL0
                LDB  @R3,RL0
                DEC  R3
          DJNZ  R2,$ XX
```

Das Register R2 wird mit 0BH geladen und ist als Schleifenzähler definiert. Die BCD-Zahl hat bezüglich dieses Wertes 22 Digits Länge. Der Wert $ 1FC1 in Register R3 zeigt auf die beiden niedrigsten Digits (zu Beginn). Das Register RH0 soll nach Anwendung des RLDB-Befehls den Digitübertrag enthalten. Er muß für den ersten Schleifendurchlauf Nullen enthalten. Die Schleife beginnt bei $ SSXX. Nachdem der Schleifenzähler R2 um 1 erniedrigt wurde, wird er auf Null überprüft und der Programmzähler auf $ SSXX gesetzt, wenn R2 nicht Null war. Dieser Sprung auf $ SSXX erfolgt relativ in einen Bereich von XX gleich −252 . . . +2, d.h. es wird ein bestimmter Wert vom momentanen Programmzählerstand nach DJNZ abgezogen. Für DJNZ wird, je nach Assembler, das Label "Loop" gesetzt, das als die Schleifeneinsprungadresse ($ SSXX) interpretiert wird. Eine BCD-Rechnung dieser Art kann mit der Anwendung des RRDB-Befehls mit geringer Programmänderung in eine Division durch 10 umgewandelt werden. Mit einem anderen Prozessor ist eine BCD-Arithmetik keinesfalls so selbstverständlich wie mit dem Z 8000.

Bei genauer Betrachtung müßte auch die arithmetische Einzelbitverschiebung in den Komplex "Rechnen" mit einbezo-

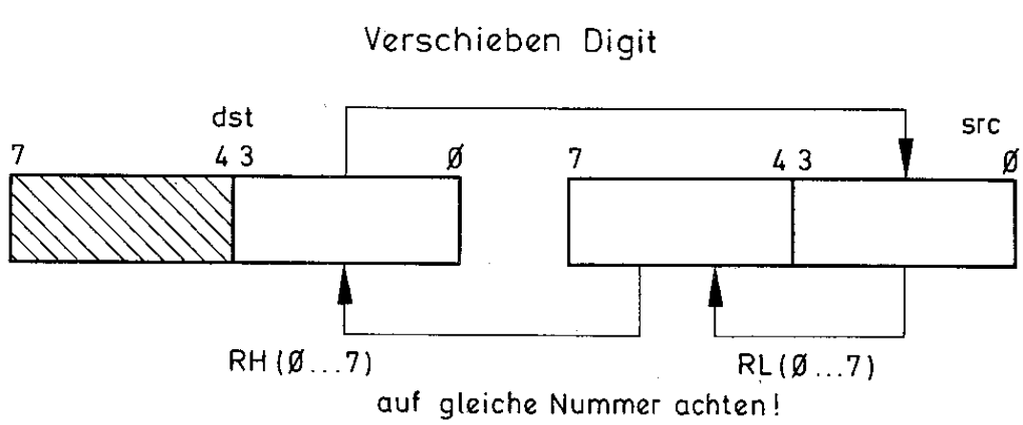


Bild 6.9

gen werden. Da die elementaren Operationen mit "Schieben" fortgesetzt werden, wird hierauf verwiesen.

Noch zwei kleine Rechenbeispiele:

Das folgende kleine Programm soll die Addition zweier beliebiger 128 bit Dualzahlen zum Ziel haben. Es werden dafür drei Register für die indirekte Adressierung der Operanden und der Zielspeicherstellen für die Ergebnisse benötigt:

R 10 → erster Operand
R 11 → zweiter Operand
R 12 → Ergebnisse in Speicherstellen

Ferner wird das Byteregister RH4 dafür benutzt, als Quellindex zu dienen. Die Anwendung des DJNZ-Befehls ist auch für dieses "Programmchen" von Vorteil. RH4 zeigt zu Beginn an, wie oft die Schleife durchlaufen werden muß. Arithmetisch wird die Operation schließlich in den Registern R0 und R1 bzw. zu Beginn in RR0 ausgeführt.

Folgendes Assemblerprogramm zeigt die Routine:

```
Zeilen-Nr.  Mnemonic  Bezeichnung              Bemerkung
1                     Addition MODULE
2                     $ NONSEGMENTED
3                     GLOBAL
4
5                     Datenfeld ARRAY [16 Worte Operanden]
6                                     [8 Worte Ergebnisfeld]
7                     Addition PROCEDURE
8
9
10          LDA       R 10 , $ %  OP.A     ! Startadresse Operand A !
11          LDA       R 11 , $ %  OP.B     ! Startadresse Operand B !
12          LDA       R 12 , $ %  ZIEL     ! Startadresse Ergebnis !
13          LDB       RH 4 , #6            ! Schleifenzähler laden !
14          LDL       RR 0 , @R 10         ! A in RR 0 mit niederwert. 32 bit !
15          ADDL      RR 0 , @R 11         ! Op. A und Op. B addieren !
16          LDL       @R 12 , RR 0         ! Ergebnis in Ziel !
17          INC       R 10 , #4            ! Startadresse inkrementieren !
18          INC       R 11 , #4            ! Startadresse inkrementieren !
19          INC       R 12 , #4            ! Start Erg. inkrementieren !
20          LD        R 0  , @R 10         ! Op. A laden (+ 16 bit) !
21          LD        R 1  , @R 11         ! Op. B laden (+ 16 bit) !
22          ADC       R 0  , R 1           ! addiere mit Übertrag !
23          LD        @R 12 , R 0          ! Ergebnis in Ziel ablegen !
24          INC       R 10 , #2
25          INC       R 11 , #2
26          INC       R 12 , #2
27          DBJNZ     RH 4 , SCHLEIFE
28                    END Addition
```

Zuerst wird, wie üblich, mit einer Initialisierung der einzusetzenden Register als Index begonnen. Das Zeichen "$ %" muß nicht angegeben werden; es ist hier als Versinnbildlichung eines Adreß-Index für die Operandenblöcke gedacht. Das Register RH 4 wird als Schleifenindex mit dem Wert 6 geladen. Sein Inhalt mit #6 ist als dezimal aufzufassen und bewirkt das sechsmalige Durchlaufen der Schleife. Für Operanden der Länge 128 bit ist dies notwendig.

Das Programm ist ansonsten sehr einfach. Die erste Addition mit ADDL betrifft die niederwertigsten 32 bit der Operandenstellen. Abgelegt wird das Ergebnis in der angegebenen Startadresse des als Index benutzten Registers R 12. Dieses wird nach dem Ladebefehl (Zeile 16) **nicht** automatisch inkrementiert. Für den Ziel-Index ist Zeile 19 zuständig. Folgend werden Operandenteile (16 bit) mit dem jeweilig vorhergehenden Übertrag (entstanden aus vorhergehender Addition) mit ADC addiert und folglich an @ R 12. Zeile 26 inkrementiert anschließend den Ziel-Index. Zeile 24 und 25 bewirkt dasselbe für die Operanden. Die Schleife umfaßt Zeile 20 . . . 27. DBJNZ beinhaltet die einfache Branch-Funktion für die Schleifensteuerung.

Eine Binärzahl der Länge 128 bit ist absolut gewaltig und findet nur in astronomischen Bereichen ihre Anwendung.

$$2^{128} = 2^{60} \cdot 2^{60} \cdot 2^{8} = 256 \cdot 16^{15} \cdot 16^{15}$$

$$\left[16^{15} = 1\,152\,921\,504\,606\,846\,876\right]$$

2^{128} ist mit etwa 42 Dezimalstellen zu schreiben. Hier muß im allgemeinen jeder Taschenrechner passen. Sicherlich dürfte hier interessant sein, wie lange die CPU daran rechnet! Insgesamt benötigt die kleine Prozedur 249 Zyklen. Setzt man für den Z 8002 eine Zykluszeit von 250 ns an, ergibt sich folgender Wert:

Prozedur für Z 8002 62,25 µs

Für den Z 8002A kann man eine Zykluszeit von 166 2/3 ns ansetzen. Es ergibt sich:

Prozedur für Z 8002A 41,5 µs

Eine Subtraktion zweier 128-bit-Operanden ist programmtechnisch sehr ähnlich aufgebaut. In diesem Fall wird ebenso mit den niederwertigen 32 bit beider Operanden begonnen, sie mit SUBL voneinander abzuziehen. Fortgesetzt wird die Operation mit dem Befehl SBC, der analog zu ADC ist. Wird bei der ersten Subtraktion bereits das S-Flag (Vorzeichen) gesetzt, ist das noch nicht maßgebend. Das S-Flag wird jedesmal neu bestimmt, so daß sich ein effektiver, wahrer Wert erst nach der letzten Teilsubtraktion der höchstwertigen Stellen ergibt. Für beide Fälle (Addition, Subtraktion) ergibt sich das automatische Einbeziehen der Übertragsbits.
Bei der Subtraktion wird zum ersten Operanden das echte Zweierkomplement des zweiten hinzugezählt.

Das zweite Beispiel ist noch etwas komplizierter. Die Aufgabe lautet:

Zwei Matrizen sollen miteinander multipliziert werden.

Damit die Matrixelemente von vorneherein eingegrenzt werden, dürfen maximale Werte von 65535 ($= 2^{16} - 1$) nicht überschritten werden. Die **erste Matrix** (A) habe 10 Zeilen mit 38 Zeilenelementen (38 Spalten mit 10 Spaltenelementen). Die **zweite Matrix** (B) habe 38 Zeilen mit 6 Zeilenelementen (6 Spalten mit 38 Spaltenelementen). Zeilen- und Spaltenelemente beider Matrizen sind demnach zahlenmäßig gleich. Die allgemeine Formel für die Berechnung eines Zeilen-/Spaltenprodukts ist:

Matrixelement aus (A · B)

$$c_{ij} = \sum_{k=1}^{S} a_{ik} \cdot b_{kj};$$

$$s = m \cdot n = 10 \cdot 6 = 60$$

$$\left[\begin{array}{ll} \text{Zeile (erste Matrix):} & i = 1,2 \ldots 10; \\ \text{Spalte (zweite Matrix):} & j = 1,2 \ldots 6; \end{array}\right]$$

Die Produktmatrix hat insgesamt 60 Elemente. Das folgende Programm für den Z 8000 muß wieder eine Schleife besitzen, die insgesamt 2280 Multiplikationen enthalten muß. Die Matrix wird folgende Zielform erlangen:

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{16} \\ c_{21} & c_{22} & & & \\ c_{31} & & & & \\ \vdots & & & c_{ij} & \\ c_{91} & & & & \\ c_{101} & \dots & & & c_{106} \end{bmatrix}$$

Wollte man hier etwa zu Fuß . . . Eine Menge Arbeit würde hier bevorstehen.

Die Zeiten kann man getrost als Vergangenheit ansehen, in der solche Matrizen noch "mit der Hand" berechnet wurden. Für einen Z 8000 ist die Berechnung dieser Matrix ein Problem einiger Millisekunden.

Man kann Überlegungen aus dem ersten Programm (Addition) für die Matrixberechnung verwerten. Zweckmäßigerweise verwendet man wieder die indirekte Adressierung. Man definiert einzelne Register als Adreßindex für den ersten Operanden, den zweiten Operanden und Ergebnisstellen im Speicher. Die Matrizen werden im Speicherwerk in einem Adreßblock beschrieben. Jedem Matrixelement entspricht eine Doppeladresse (für zwei Bytes).

Die Summenformel für die allgemeine Matrixmultiplikation sagt aus, daß alle Produkte aus einer Zeilen-/Spaltenmultiplikation addiert bzw. subtrahiert werden müssen. Treten negative Matrixelemente auf, ist der Wertebereich N für ein Matrixelement

$$+32767 \geqslant N \geqslant -32768.$$

Das MSB gilt als Vorzeichenbit und ist gleich 1, wenn das Matrixelement negativ ist. Es ergibt sich somit

$$c_{11} = \pm a_{11}\, b_{11} \pm a_{12}\, b_{21} \dots \pm a_{1\,38}\, b_{38\,1}$$

Damit sind prinzipiell die einzelnen Phasen für das Programm genannt. Sie bestehen aus

- Initialisierung (Adreßindex für Elemente, Schleifenindex)
- Multiplikation Zeile/Spalte
- Zweierkomplement-Addition
- Schleifensprung nach Berechnung eines Ergebniselemente c_{ij} (Zeilenzahl von erster Matrix als Schleifenindex)

Mit einer Schleife ist es allerdings nicht getan. Insgesamt treten drei Schleifen auf, die allè mit DBJNZ behandelt werden, weil diese einfache Branch-Funktion alles beinhaltet, was man braucht. Die Einsprungadresse für die Schleife ist immer dieselbe. Für die Branch-Funktion werden drei Byteregister als Zählerindex benützt.

RL 6 für die Zeilen/Spaltenprodukte
RH 6 für die Zeilenzahl der C-Matrix
RH 7 für die Spaltenzahl der C-Matrix

Das folgende aufgelistete Programm ist sicherlich nicht mehr ganz so einfach wie die bisherigen. Man stelle sich jedoch nur vor, diese Aufgabe mit dem 8080 zu lösen. Ernste Probleme würden nicht auf sich warten lassen. Mit dem Z 8000 ist das einfach kein Problem. Es bleiben immer noch einige Register ungenutzt.

Die Einsprungadresse aller Schleifen ist die Zeile 18 mit RESFLG. Nach der zweiten Verzweigung wird eine neue Startadresse für die nächste Spaltenreihe der Matrix B geladen. Dies ist deshalb so gedacht, weil die erste Spalte von Matrix B grundsätzlich erst mit allen Zeilen der Matrix A multipliziert wird. Die letzte Verzweigung bedingt die Abfrage der noch zu bearbeitenden Spaltenzahl von Matrix B.

Auf eine Eingrenzung muß noch aufmerksam gemacht werden. In dem Ad-

Zeilen-Nr.	Mnemonic	Bezeichnung	Bemerkung
1		Matrix MODULE	
2		$ NONSEGMENTED	
3		GLOBAL	
4			
5		Datenfeld ARRAY [6 x 38 Worte]	
6			
7		Multiplikation PROCEDURE	
8			
9			
10	LDA	R 10 , $ % Matrix A	! a_{11} !
11	LDA	R 11 , $ % Matrix B	! b_{11} !
12	LDA	R 12 , $ % Produkt	
13	LDL	$ % Produkt , #0	
14	LDA	R 13 , $ % Zielmatrix C	! c_{11} !
15	LDB	RL 6 , #38	! Zeilenprodukte !
16	LDB	RH 6, #10	! Zeilen C-Matrix !
17	LDB	RH 7, #6	! Spalten C-Matrix !
SCHLEIFE → 18	RESFLG		
19	LD	R 0 , @R 10	! laden Element von A !
20	MULT	RR 0, @R 11	
21	ADDL	RR 0, @R 12	! Addition $a_{ik} \cdot b_{kj}$!
22	LDL @	R 12 , RR 0	! ablegen Teilsumme !
23	INC	R 10 , #2	! Operandenzeiger inkrem. !
24	INC	R 11 , #2	! Zeiger f. Matrix B inkrem. !
25	DBJNZ	RL 6 , SCHLEIFE	
26	LDL @	R 13 , @R 12	! K oeff. Zielmatrix c_{11}, c_{21} !
27	INC	R 13 , #4	! Zeiger für Zielmatrix inkrem. !
28	LDB	RL 6 , #38	! Zeilenprodukte neu f. Schleife !
29	LDA	R 11 , Matrix B	! Spalte b_{11} laden der Adresse !
30	DBJNZ	RH 6, SCHLEIFE	
31	LDB	RH 6 , #10	
32	LDA	R 10 , Matrix A	! Adresse für Zeile a_{11} !
33	LDA	R 11 , Matrix +1B	! Adresse für neue Spalte von b !
34	DBJNZ	RH 7, SCHLEIFE	
35		END Multiplikation	

ditionsschritt (Zeile 21) wird das letzte Spalten-/Zeilenprodukt (Zeile zuvor) mit der Teilsumme addiert. Es ist darauf zu achten, daß nach dem letzten Schleifendurchlauf kein Überlauf von RR 0 entsteht, d.h. ein Zielmatrix-Element c_{ij} darf zahlenmäßig nicht 2^{31} (bei Vorzeichenbit) übersteigen. Ansonsten wäre das Ergebnis falsch.

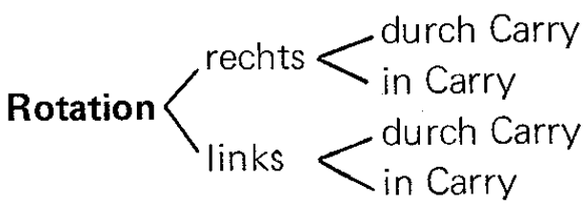
Auch in diesem Programm muß für die Inkrementierung der Zeiger gesorgt sein (Zeile 23, 24 und 27).

Schieben:

In Abschnitt 3.4.6 wurde ein allgemeiner Überblick über die Schiebe- und Rotationsoperationen gegeben, die mit der CPU durchführbar sind. Im einzelnen wird hier noch weiter darauf eingegangen.

Zwischen schieben und rotieren besteht ein nennenswerter Unterschied. Bei einer Schiebeoperation werden nach dem Prinzip des Fallens eng und hochkantig aneinander stehender Dominosteine (stellvertretend für einzelne Registerstellen) in einer Richtung Bits verschoben. Je nach Fall, links oder rechts, geht dabei die Information verloren, weil sie sprichwörtlich hinausgeschoben wird. Es verbleiben im Register entweder lauter Nullen oder Einsen. Bei einer Rotationsoperation ist der Vorgang nicht anders, was die Verschiebung betrifft. Der Unterschied liegt nur darin, daß die Information nicht verloren geht. Die Positionsänderung der einzelnen Stellen bewirkt nach dem Austritt an einem Registerende unmittelbar den Eintritt der Parität am anderen Registerende, was gleichsam einem Ring nahe kommt. Die englische Bezeichnung "barrel shifter" ist für ein Rotationsschieberegister symtomatisch. Aus einem dichten Faß geht nichts verloren. Bei Rotationsoperationen bildet ein CPU-Register gleichsam diesen Ring durch eine quasi äußere Verbindung zwischen MSB und LSB des Registers. Bemerkenswert ist überdies, daß jedes Allzweckregister (Byte- oder Wortregister) dafür verwendet werden kann, – auch ein Stapelzeiger.

Sieht man einmal von den bereits behandelten Digit-Rotationen RLDB und RRDB ab, bleiben vier Arten einer Einzelbit-Rotation:



Der Unterschied im Operationstyp "durch Carry" mit RL(R)C und "in Carry" mit RL(R) zeigt sich in der Länge des Verschiebeweges. Durch das Carry-Flag besagt z.B. bei einem Wortregister die 17. Stelle. Tritt eine Eins am MSB des Registers in das Carry ein, das zuvor Null war, tritt diese Null an die Stelle des LSB. Erst der zweite Schiebeschritt transferiert die Eins aus dem Carry in das LSB des Registers. **Bild 6.10** zeigt die Darstellung der Rotation durch das Carry-Flag. Es sind hierfür keine Doppelwort-Register einsetzbar. Prinzipiell gilt RL(R) auch für Register 15, das jedoch hierzu nicht mißbraucht werden sollte.

Bild 6.11 zeigt die übliche Rotation in das Carry. Liegt im MSB eine Eins eingeschrieben und ist das Carry-Flag auf Null gesetzt, wird nach einmaliger Linksverschiebung diese Eins im LSB des Registers sein und das Carry-Flag auf Eins stehen.

Für die Rotation gilt im allgemeinen die Ausführung von Einzelverschiebungen ein- oder zweimal. Für den Assembler gilt ferner, daß eine Einzelverschiebung nicht mit # 1 angegeben werden muß, vergleichbar mit INC oder DEC bei einer Veränderung um 1. Bei zweimaliger Ausführung muß die Verschiebung mit # 2 hinter dem Komma angegeben werden. Ferner ist es überflüssig, einen Wert im Register zu bezeichnen. Es wird als Ziel immer ein Register angegeben, in dem der zu rotierende Wert steht. Ist eine Rotationsoperation ausgeführt worden, wird darauf logischerweise eine Branch-Operation (JP, JR) folgen. Damit ist leicht angedeutet, daß mit Rotationen neben einer Arithmetik gezielte zyklische Vorgänge auszulösen sind. Man setzt beispielsweise in ein Wortregister den Wert 8000H ein, folglich 1000|0000|0000|0000, und läßt den Inhalt links rotieren. Nach jedem Schritt wird C über JR abgefragt und gesprungen, wenn es gesetzt war. Der Vorteil dabei ist, daß dieser Vorgang nur aus zwei Befehlen besteht, RL und JP(JR), und beliebig oft verwendet werden

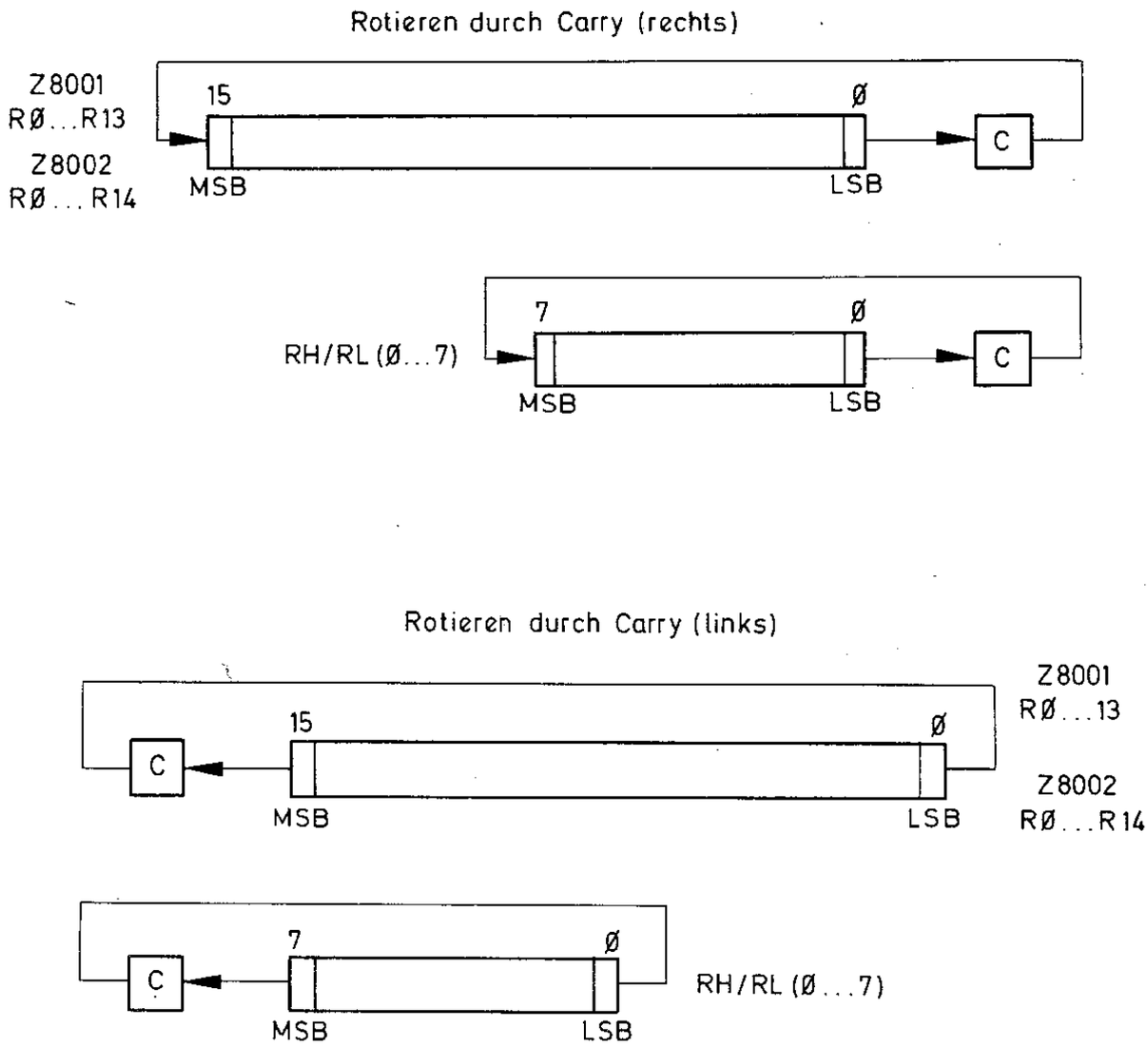
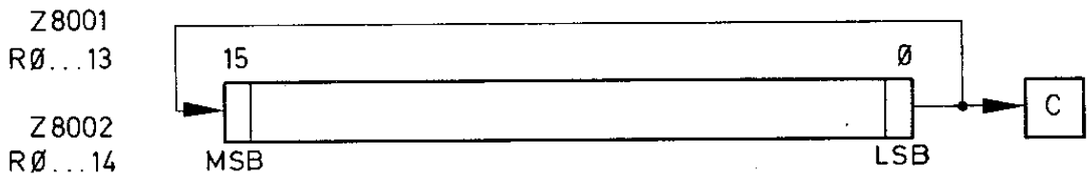
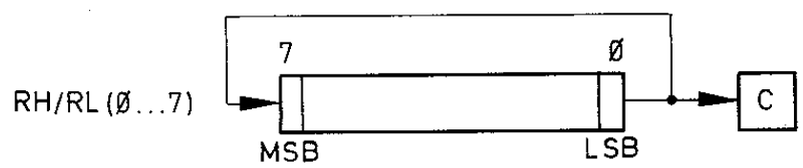


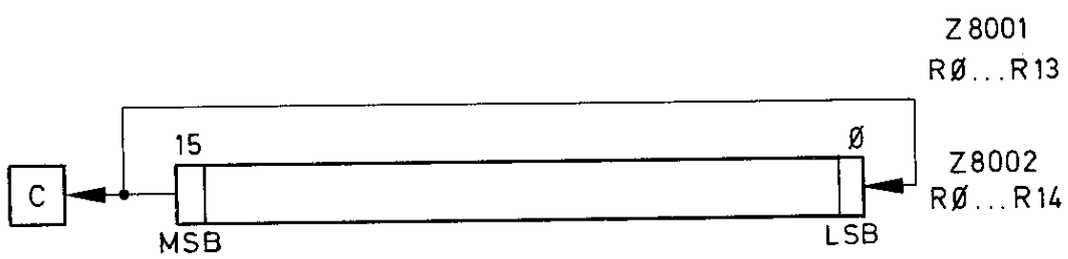
Bild 6.10

kann, ohne einen Index neu setzen zu müssen. 15 Vorgänge könnten mit einer Serviceroutine bedient werden, wenn von der Eins abgesehen wird.

Für Schiebeoperationen gilt der Typus **"logisch", "dynamisch logisch", "arithmetisch"** und **"dynamisch arithmetisch".** Schiebeoperationen sind Doppelwortbefehle. Die arithmetische und logische Schiebeoperation bezeichnet als Ziel ein Register, dessen Inhalt verschoben wird, und als Quelle einen unmittelbaren Wert, der die Anzahl der Schiebeschritte festlegt. Für die Befehle SLA und SLL (Schieberichtung links) gilt die schematische Darstellung der Operation, wie sie **Bild 6.12** zeigt. Ob arithmetisch oder logisch, immer wird rechts am LSB des Registers eine Null eingeschoben. Dies ist grundsätzlich so. Für die Befehle mit einer **Schieberichtung nach rechts gelten unterschiedliche Konventionen. Bild 6.13** zeigt die Schematik für den Befehl SRA. Es ist in diesem Fall abhängig, ob das MSB des Inhalts eine Eins oder eine Null ist. Ist das MSB gleich 1, wird immer eine 1 eingeschoben. **Bild 6.14** zeigt hingegen die Schematik für den Befehl SRL. Hier ist es unabhängig,







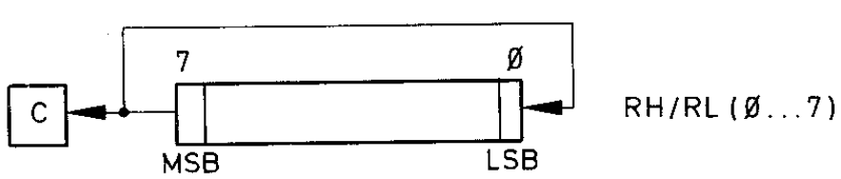


Bild 6.11

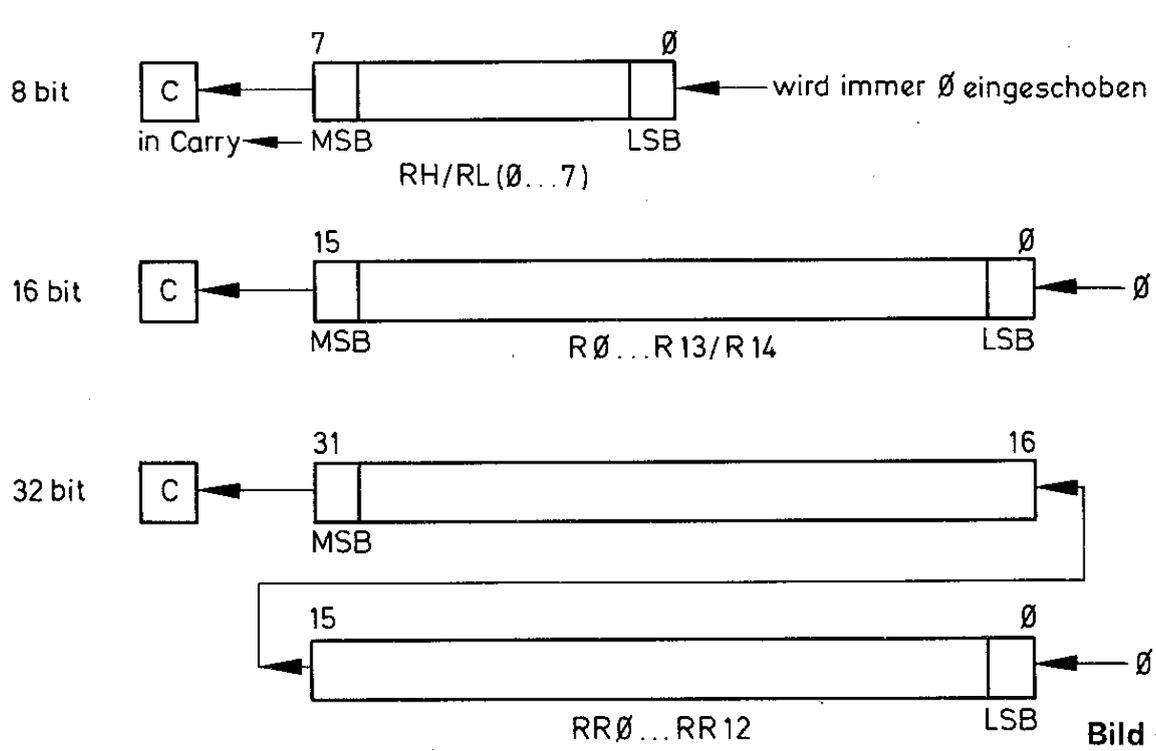


Bild 6.12

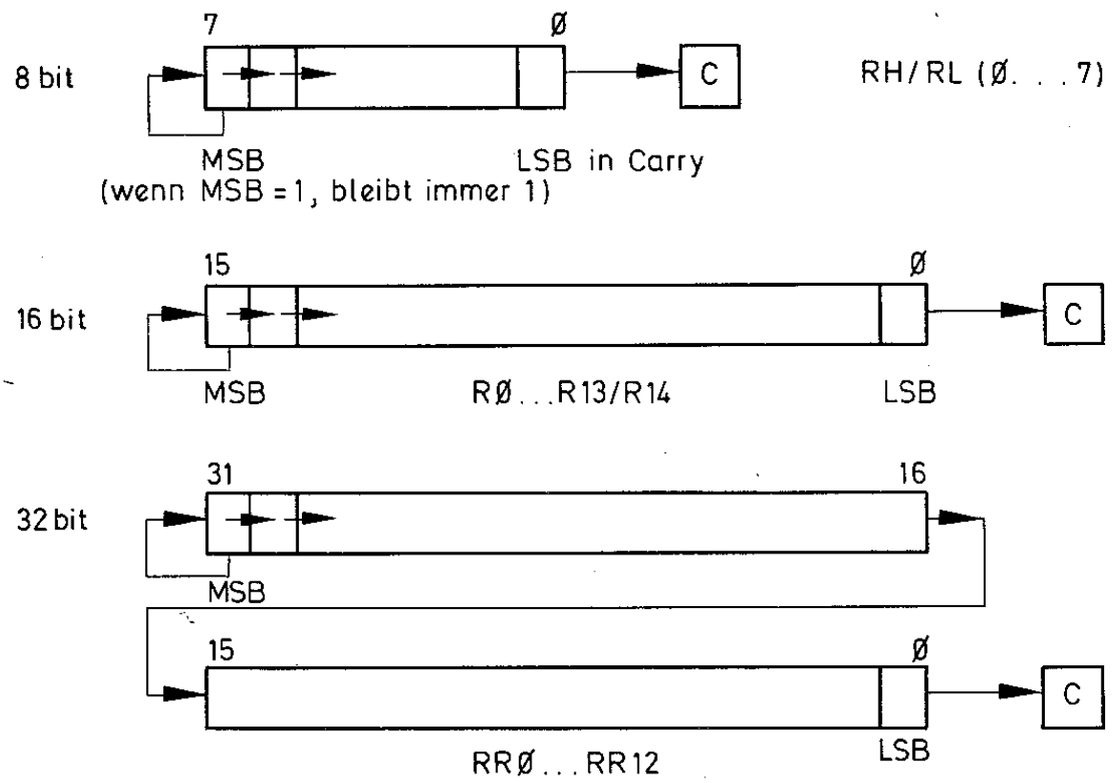
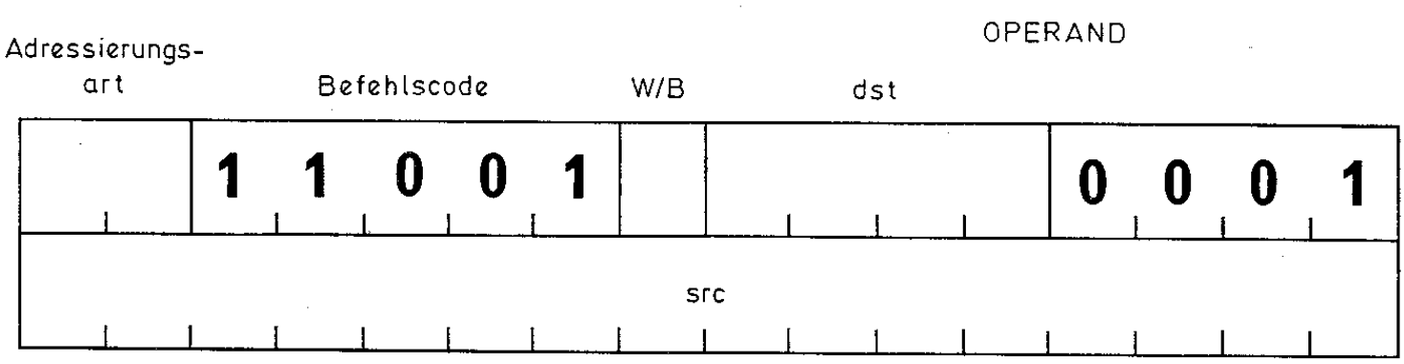


Bild 6.13

welchen Wert das MSB einnimmt. Bei jeder Schiebeoperation wird eine Null nachgeschoben. In dem 16-bit-src-Feld wird der direkte, binäre Wert eingeschrieben, wie oft eine Einzelbitverschiebung durchzuführen ist. Für den Befehl SLAL können maximal 32 (= 10000 binär) Verschiebungen durchgeführt werden. Das bedeutet, daß eine Eins im LSB eines Doppelwortregisters im Carry-Flag wieder zum Stillstand kommt, wenn die Operation zu Ende geführt worden ist. Gibt man den Wert Null als Anzahl an, verändert sich im Inhalt des betroffenen Registers nichts. Es werden nur die betroffenen Flags geändert, die für den Befehlstyp und die gegebene Konfiguration des Inhalts gesetzt sein müssen.

Der Opcode des Befehls SLL ist für die Wortausführung mit 1 an W/B zu füllen (für Byte, → W/B → 0). Für die Langwortausführung wird das 4 bit Feld rechts statt mit 0001, mit 0101 ersetzt (plus W/B = 1)
Für das Feld der Adressierungsart gilt generell 10



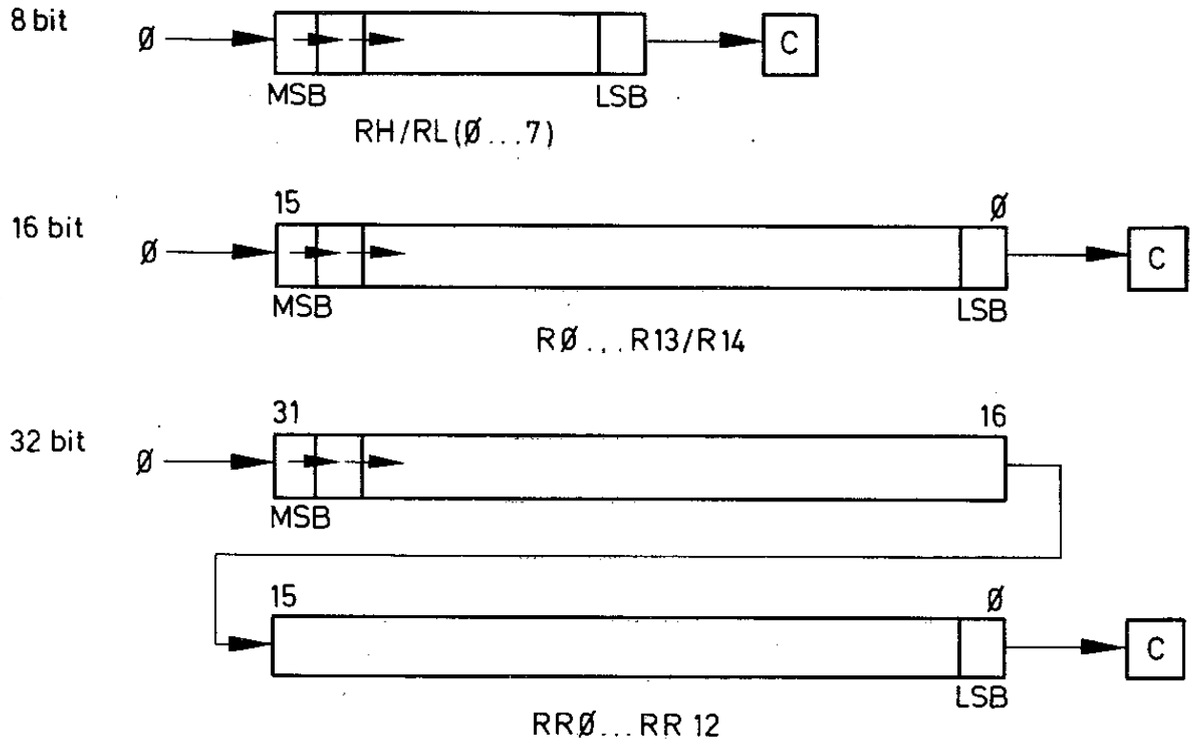


Bild 6.14

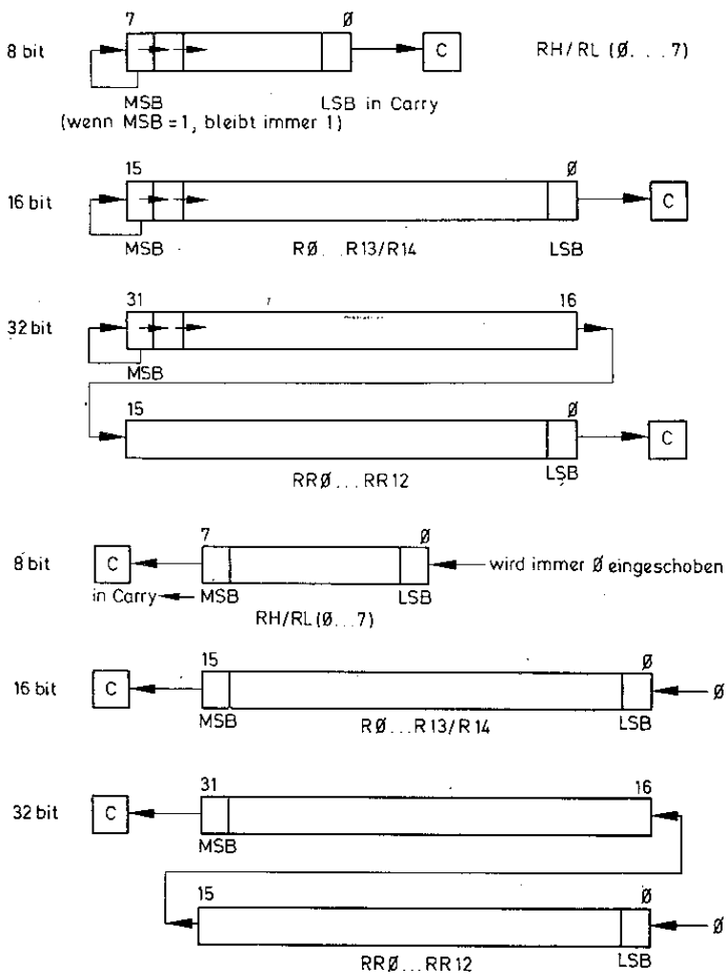


Bild 6.15

Die Schiebeoperationen gipfeln gleichsam in der dynamischen Ausführung der arithmetischen und logischen Verschiebung. Die Verschiebeanzahl steht für die betroffenen Befehle SDA und SDL in einem beliebigen Register und beinhalten die Vorzeichenkonvention. Der zu verschiebende Operand kann das übliche Formatspektrum von Byte bis Doppelwort umfassen. Es ist Rechts- oder Linksverschiebung möglich. Ein **negativer Wert** gilt als **Rechtsverschiebung,** ein **positiver Wert als Linksverschiebung.** In der folgenden Tabelle sind für die beiden Befehlstypen und ihrer Formate die Verschiebegrenzen angegeben.

links	**dyn. Verschiebebefehl**	**rechts**
+ 8	SDAB , SDAL	– 8
+ 16 . . .	SDA , SDL	– 16
+ 32 . . .	SDAL , SDLL	– 32

Der Quelloperand bezeichnet einen 16-bit-Zweierkomplement-Wert. Für eine Rechtsverschiebung um zehn Stellen ergibt sich für src der Inhalt FFF6H (≙ 1111|1111|1111|0110). Für die dynamischen Schiebeoperationen gilt überdies die gleiche Vorschrift, wie sie bereits in den Bildern 6.12 . . . 6.14 dargestellt wurde. **Bild 6.15** stellt die Schematik für den Befehl SDA zusammen. **Bild 6.16** zeigt die Schematik für den Befehl SDL.

Transferieren:

Der Bearbeitung von Daten bzw. Operanden geht häufig ein Transfer an die gewünschte Stelle voraus. Ein Transfer von Operanden aus Registern oder Speicherplätzen bedarf nicht selten größeren

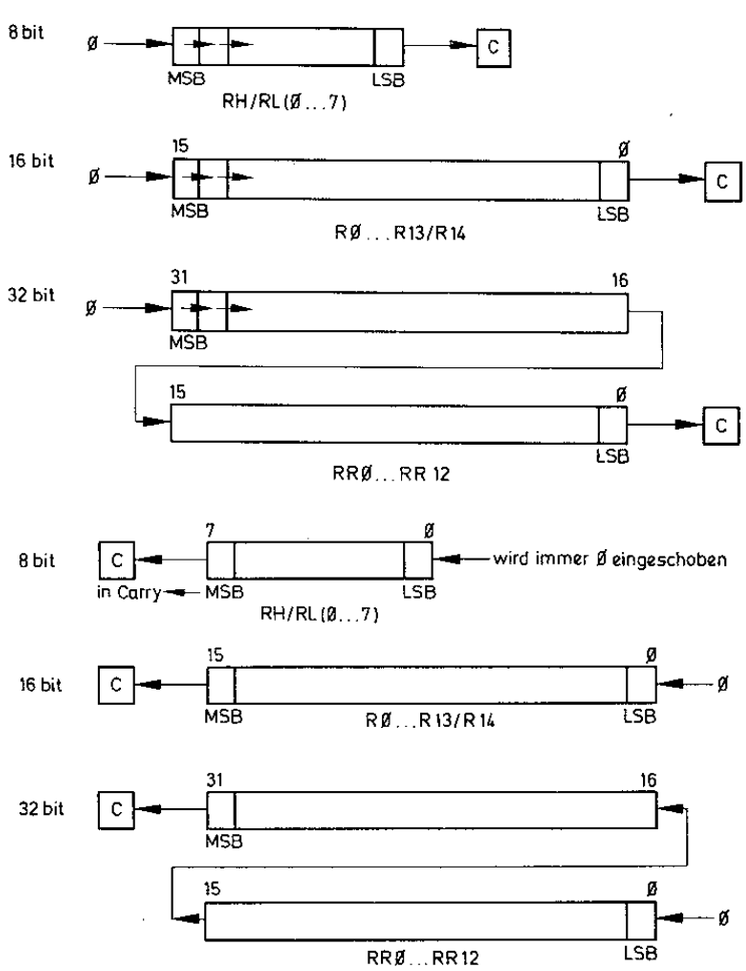


Bild 6.16

Zeitaufwands. Die Vorbereitung von Speicherbereichen mit Datenbankcharakter gehört für größere Anlagen einfach zur Routine. Die überwiegende Zahl von Programmen sind datenintensiv, d.h. es wird mit sehr vielen Operanden hantiert. Die Notwendigkeit einer einfachen Behandlung von Zeichenketten, Blockdaten usw. trägt der Befehlssatz der Z 8000-CPU Rechnung. In Abschnitt 3.4.7 wird ein Überblick über die Transfer-Operationen der CPU gegeben. Kompromisse wurden hier nicht gemacht. Der Anwender merkt sehr schnell, welche Vereinfachungen für ihn bereit stehen. Ein typischer Transfer-Befehl ist **LDIR.** Der Befehl transferiert eine auf das Byte genau definierbare Zahl von Daten beliebiger Art (Programmcode oder Operanden), die einen maximalen Umfang von 65536 Bytes oder 32768 Worten haben dürfen. In **Bild 6.17** ist in grafischer Form die Wirkung des Befehls mit einer speziellen Registerdefinition dargestellt. Die Transfer-Operation

```
LDIR   @R 11 , @R 10 , R 14
```

benützt ausschließlich die indirekte Adressierungsart. Das Register R 10 dient als Quelladreßzeiger, R 11 dient als Zieladreßzeiger. Beide Register adressieren zwei Bereiche im Speicherwerk, die jeweils mit 4100H und 5100H beginnen. Das Register R 14 definiert die besagte Anzahl von (in diesem Fall) Worten, die transferiert werden sollen. Zuerst wird die Adresse für ein zu holendes Quellwort ausgegeben (% 4100). Das Zwischenspiel ist das Register R 0 der CPU. Anschließend wird die Adresse an den Blockanfang ausgegeben (% 5100), an den das Wort eingeschrieben werden soll. Quell- und Zieladresse werden jeweils um 2 Stellen inkrementiert (% 4102, % 5102) und der Zählerindex um 1 dekrementiert (% FF). Es wird mit der Quelladreßausgabe fortgesetzt. Die Befehlsausführung ist genau an dieser Stelle unterbrechbar.

Ein Datentransfer besteht nun nicht durchweg aus solchen umfangreichen Veränderungen. Veränderungen sind häufig im kleinen Maß und insbesondere in den CPU-Registern nötig, um für die nächste Aufgabe sichere Bedingungen zu schaffen. PUSH und POP sind zueinander inverse Befehle, die allgemein häufig anzuwenden sind. **Bild 6.18** zeigt den typischen Fall eines Transfers eines Registerinhalts in (auf) den Stapel. Dies geschieht mit dem Befehl

```
PUSH   @ R 10 , R 4
```

wie gezeigt. R 10 ist in diesem Fall als Stapelzeiger benützt, der **vor der Adreßausgabe** von % 4102 auf % 4100 erniedrigt wird. R 4 ist das Register, dessen Inhalt abgelegt werden soll. Wie man daraus erkennen kann, deuten hohe Stapelzeiger-Adressen auf einen wenig gefüllten Stapelspeicher hin. Die Zeigeradresse sinkt beim Füllen des Stapels. Die Umkehrung dieser Operation ist

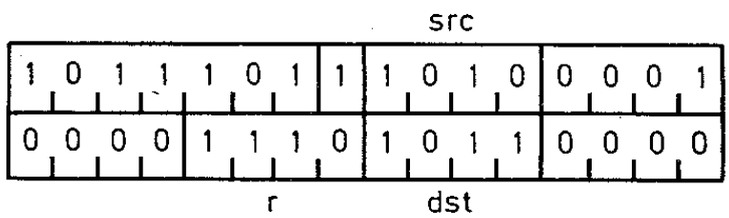
```
POP    R 4, @ R 10
```

Die Ausführung bewirkt genau den umgekehrten Fall des PUSH-Befehls. Wiederum wird **vor der Adreßausgabe** der Stapelzeiger R 10 um 2 Stellen inkrementiert, bevor der Stapelplatz ausgelesen und in R 4 transferiert wird. **Bild 6.19** zeigt die Grafik der Ausführung von POP.

Die Befehlskombination PUSH und POP kann selbstverständlich auch für Speicherplätze angewandt werden. In diesem Fall wird ausschließlich indirekt adressiert. Mit

```
PUSH   @R 10 , @R 4
```

wird der durch R 4 adressierte Speicherplatz auf den Stapel transferiert. Der Stapelzeiger R 10 wird um 2 Stellen erniedrigt, bevor er die Stelle im Stapel adressiert. Wichtig ist hier wieder, wie schon bei LDIR, daß R 0 als Zwischen-

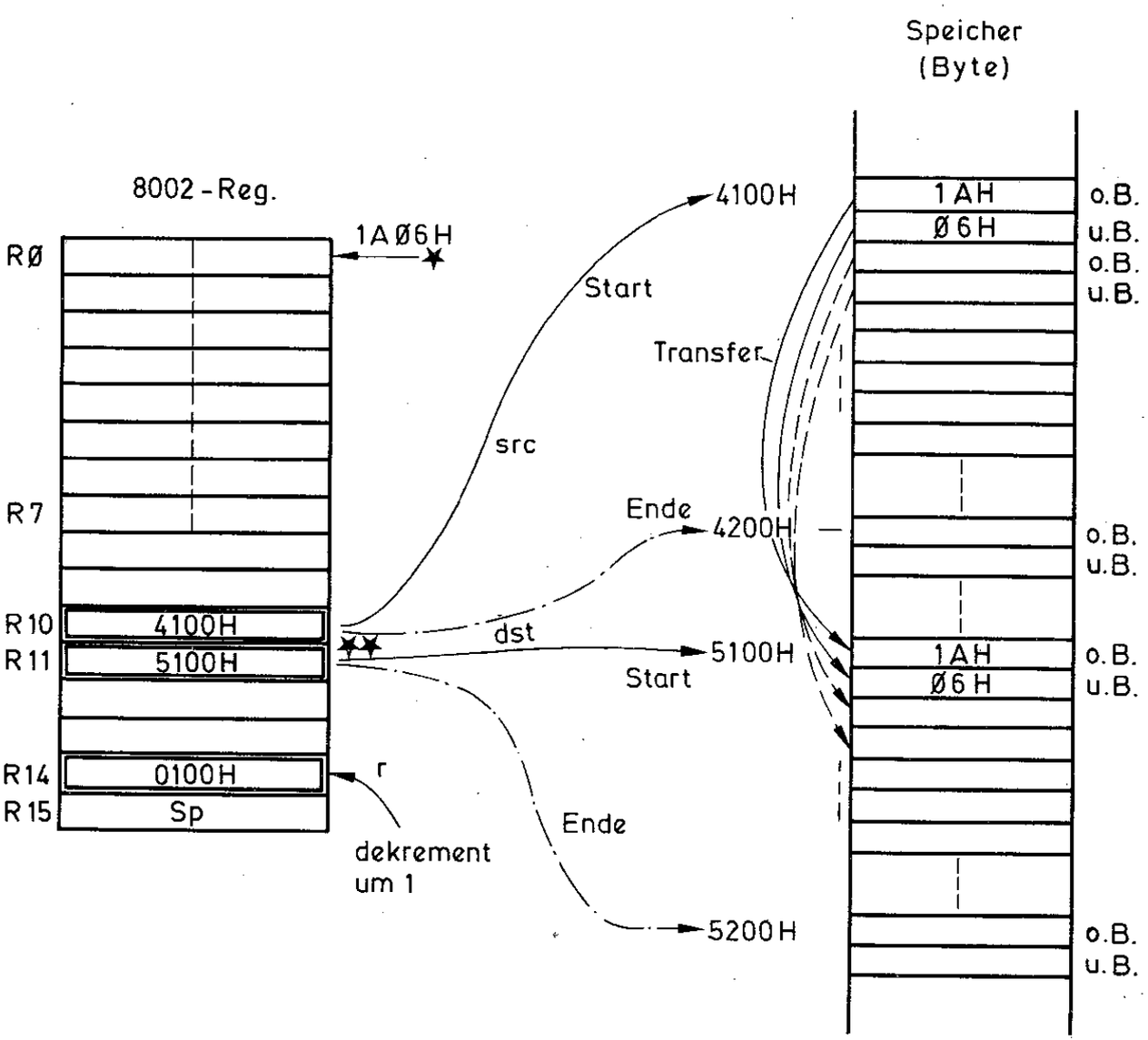


Laden mit Inkrement
(wiederholend)

r = R14; dst = R11; src = RØ

LDIR @ R11, @ RØ, R14

automatisch inkrementierend



✶ Transferzwischenspeicher in RØ

✶✶ werden um 2 Stellen inkrementiert

Bild 6.17

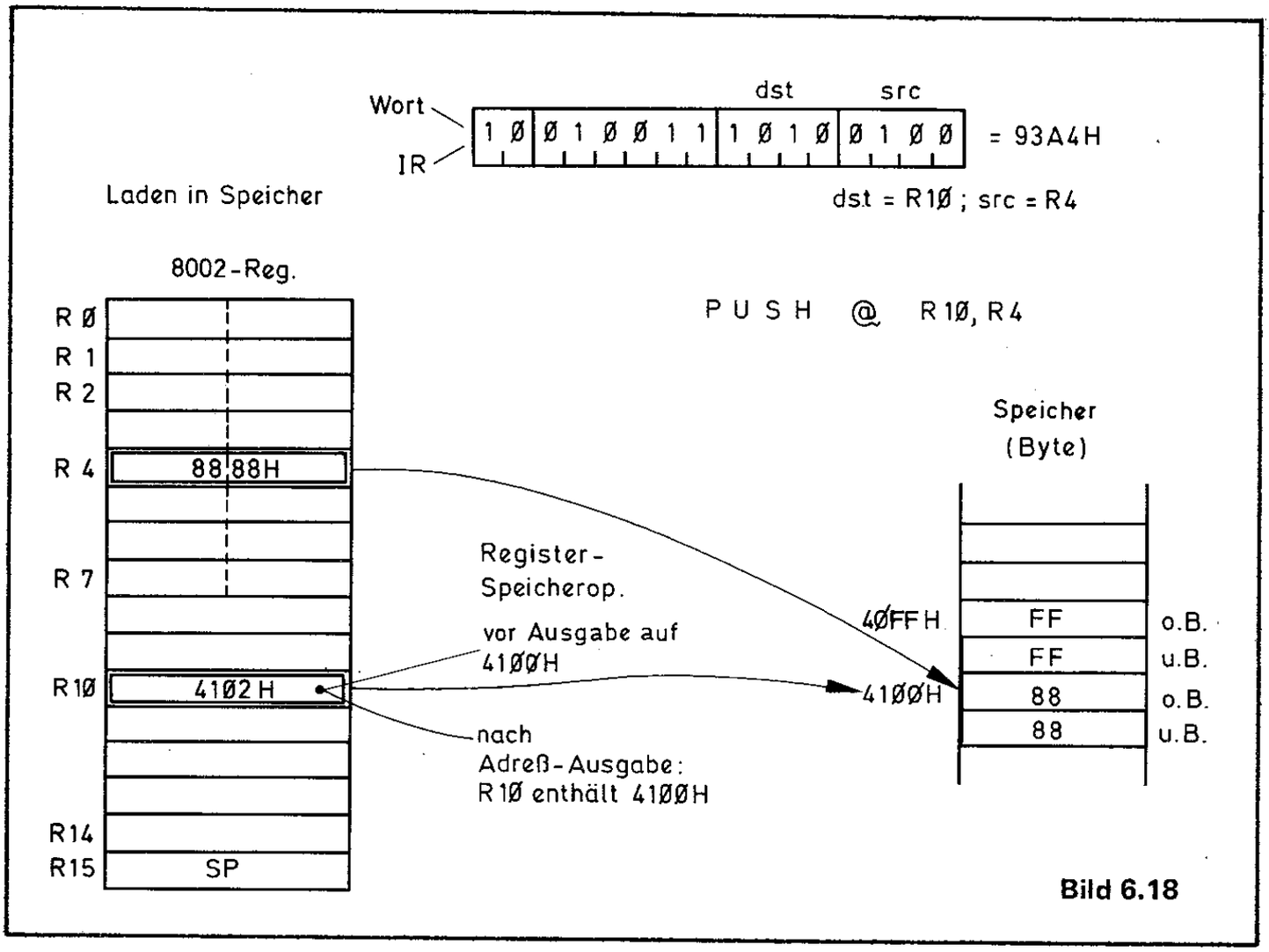


Bild 6.18

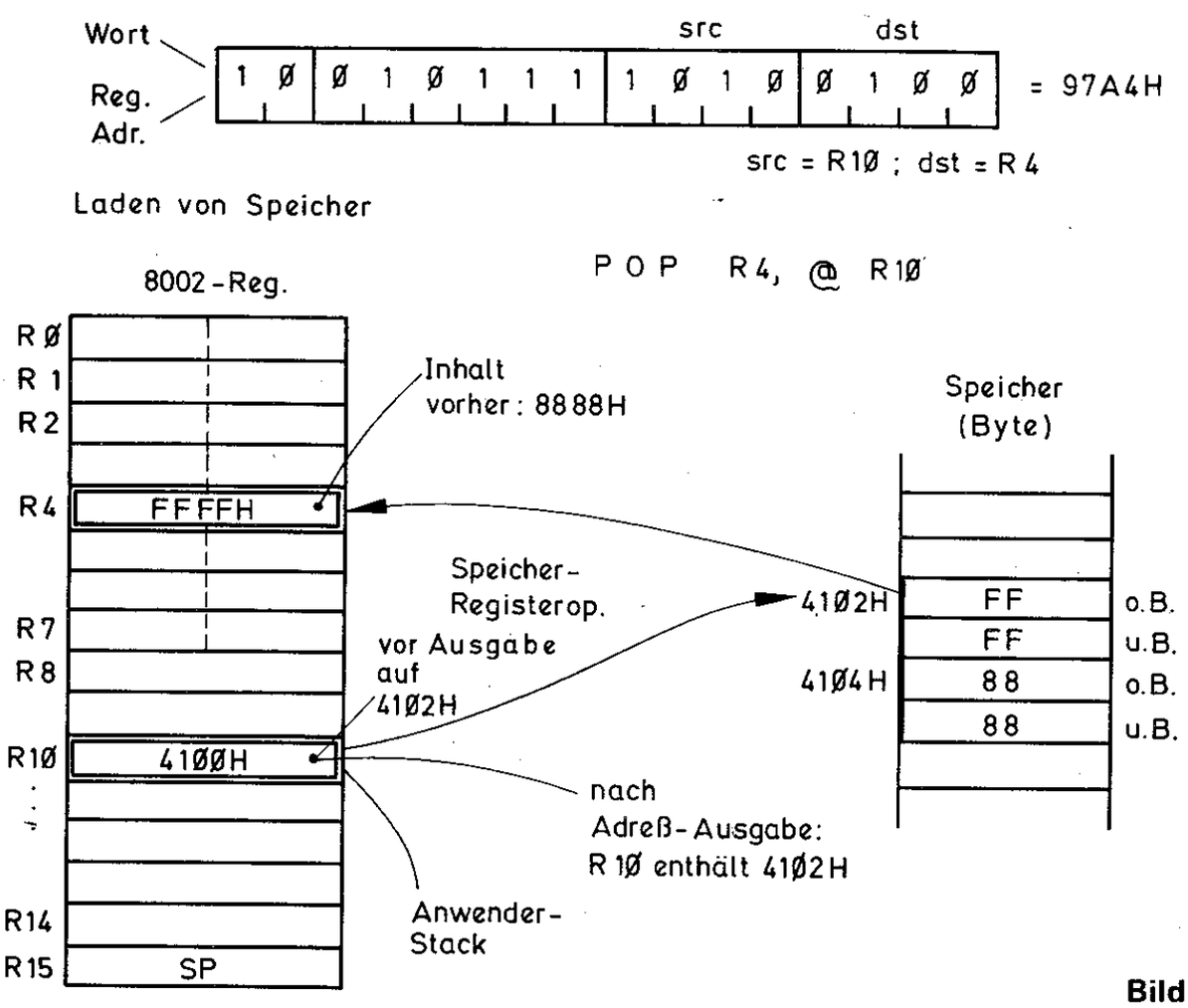


Bild 6.19

speicher verwendet wird. **Bild 6.20** zeigt die Ausführung dieser PUSH-Variante. Die unmittelbare Umkehrung zu diesem Befehl erfolgt durch

POP @R 4 , @R 10

Der auf % 4100 stehende Stapelzeiger R 10 wird **vor der Adreßausgabe** auf % 4102 erhöht. Der Stapelplatz wird ausgelesen und wieder in R 0 zwischengespeichert. Sodann wird der Zielzeiger aktiviert und an der ausgegebenen Datenspeicheradresse % 5100 der Wert FFFFH eingeschrieben. **Bild 6.21** zeigt die Grafik des Befehls POP mit indirekter Adressierung.

Eigentlich hat diese PUSH-/POP-Variante etwas vom Thema abgelenkt, das kurz zuvor auf das Transferieren von Registerinhalte abgezielt war. Die Vorstellung des folgenden Befehls trifft hier genau ins Schwarze bezüglich des Themas. Der Befehl LDM erledigt diverse Registerrettprozeduren exzellent. Schreibt man beispielsweise

LDM @R 12 , R 0 #11

drückt die Zahl #11 aus, daß die Inhalte von Register R 0 bis einschließlich R 10 in den Speicher transferiert werden. Das Register R 12 dient als Zeigerindex des gewünschten Speicherbereichs. Nach jedem transferierten Byte (oberes, unteres) des Registerwortes wird der Wert von R 12 um 1 inkrementiert. **Bild 6.22** zeigt den LDM-Befehl grafisch. Der Start bei % 4100 transferiert das obere Byte von R 0; abgeschlossen wird die Operation mit dem Transfer des unteren Byte an Adresse % 4115. Der LDM-Befehl erlaubt nicht nur den Transfer von Register in Speicher. Es ist ebenso die Richtung von Speicher in Register möglich, wenn der entsprechende Opcode verwendet wird. Als Ergänzung zu Bild 6.22 sei noch das Gegenstück des

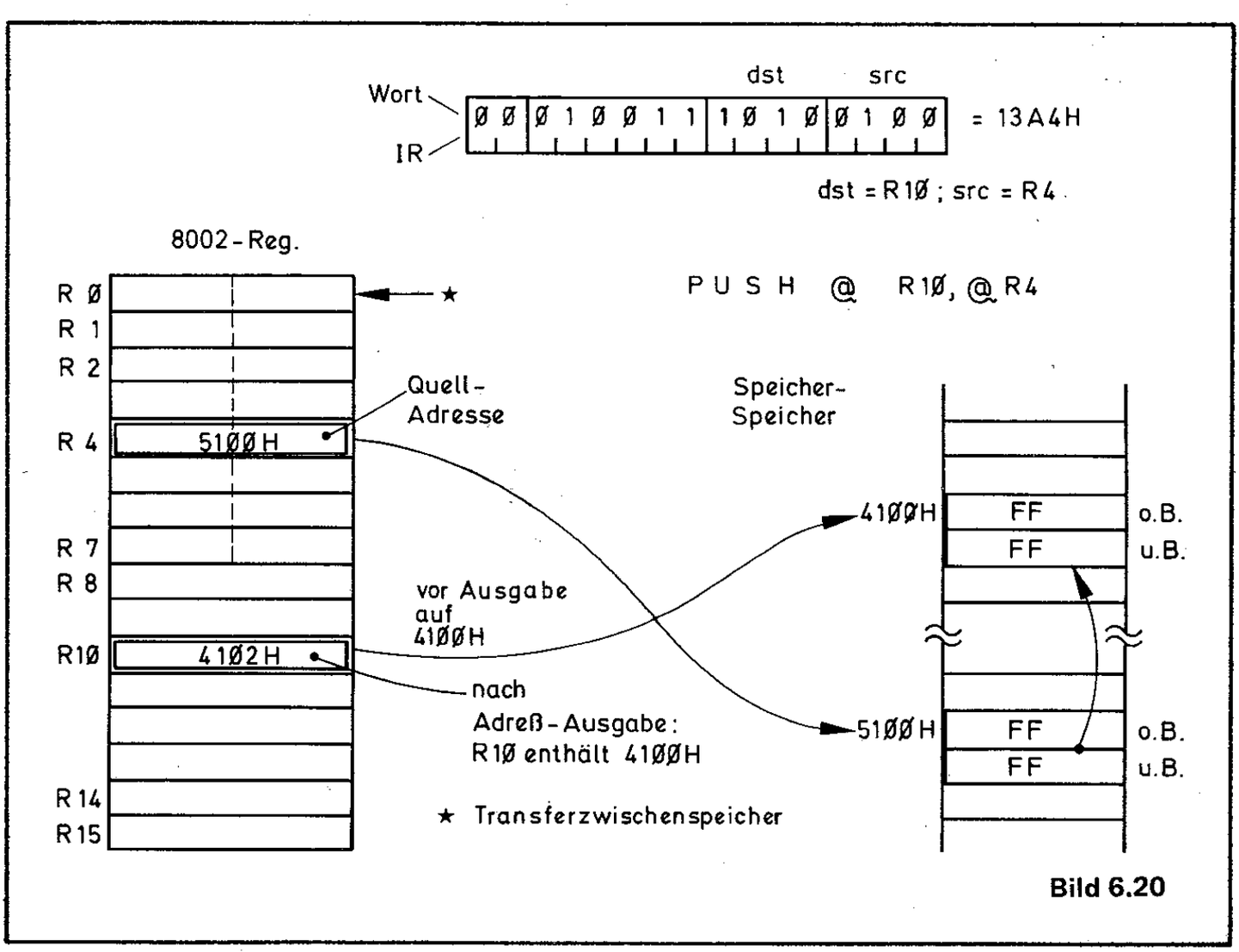


Bild 6.20

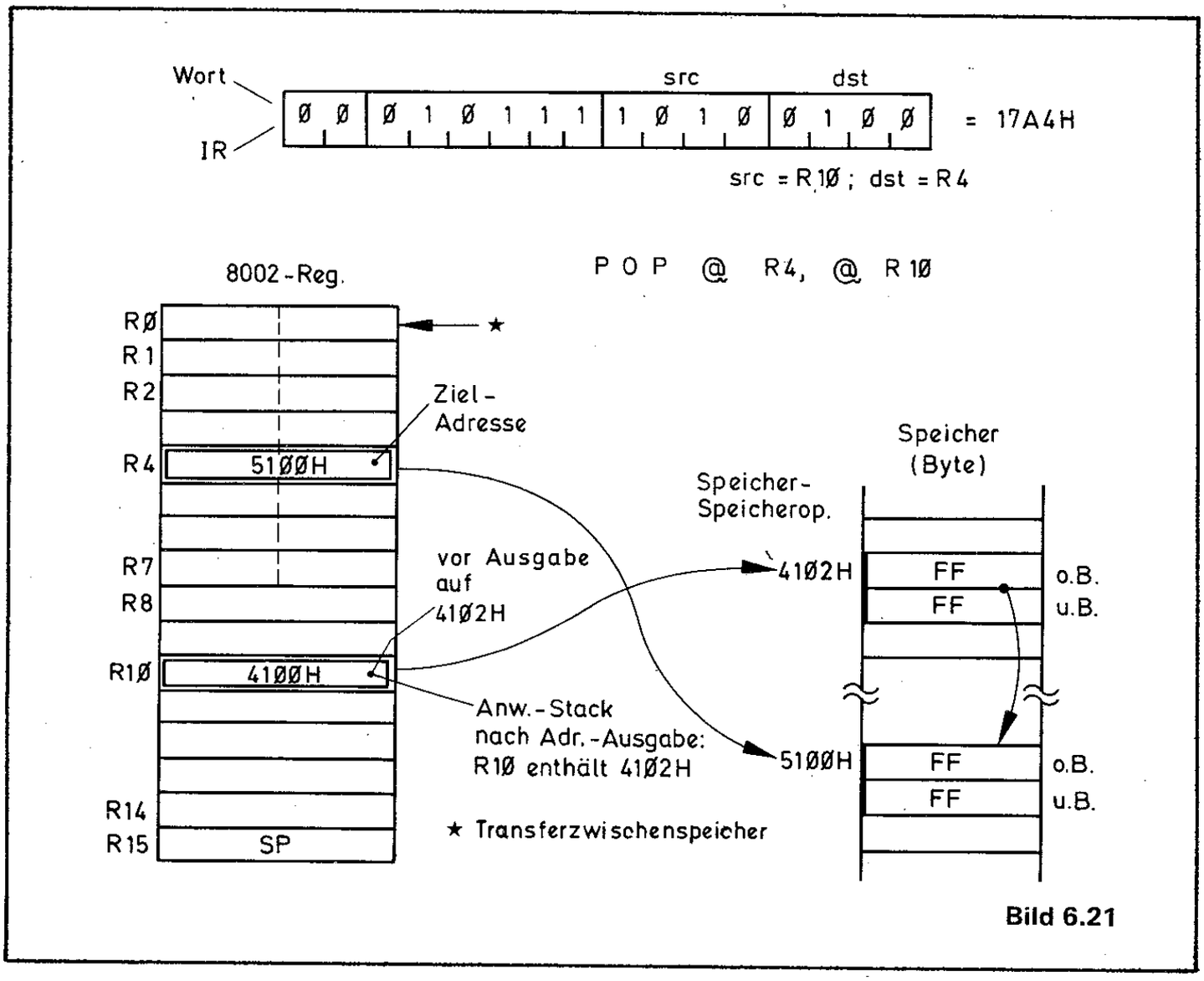


Bild 6.21

Befehls-Opcodes dargestellt: Transfer von Speicher in Register.

Mode	0 1 1 1 0 0	src	0 0 0 1
0 0 0 0	dst	0 0 0 0	num

Es kann überdies frei definiert werden, ab welchem Register der Transfer erfolgen soll. In dem obigen Beispiel wird mit R 0 begonnen. Man kann auch erst bei R 2 beginnen. Schreibt man als Quelle

R 0 # 16 ,

wird, angefangen von R 0, bis zum Stapelzeiger transferiert.

Es sei hier nur nebenbei bemerkt, daß die 8086-CPU einen Befehl dieser Art nicht kennt. Der einzige 68000 von Motorola besitzt mit dem Befehl MOVEM eine gleichwertige Alternative. Bei hochgradig registerorientierten Maschinen gehört solch ein Befehl auch unbedingt dazu.
Damit wurde der Bereich Transfer kurz angesprochen.

Segmentsprung.

Im Prinzip wurde der Segmentsprung in seiner syntaktischen Form mit dem Assembler in Abschnitt 3.3 behandelt. In Abschnitt 2.3 wurde erwähnt, daß ein Segmentübertritt in erster Linie mit "CALL Procedure" abgewickelt wird. Diese bereits bekannten Dinge werden im folgenden noch etwas detaillierter dargestellt. Mit CALL wird begonnen.

CALL und CALR sind die Befehle, die unmittelbar den Programmzähler (PC) des Z 8000 betreffen. CALL ist kein besonders Z 8000 typischer Befehl. CALL ist gleichlautend für die 8-bit-CPUs des Typs Z 80, 8080 und 8085. Er lautet JSR (Jump to Subroutine) für den 6502

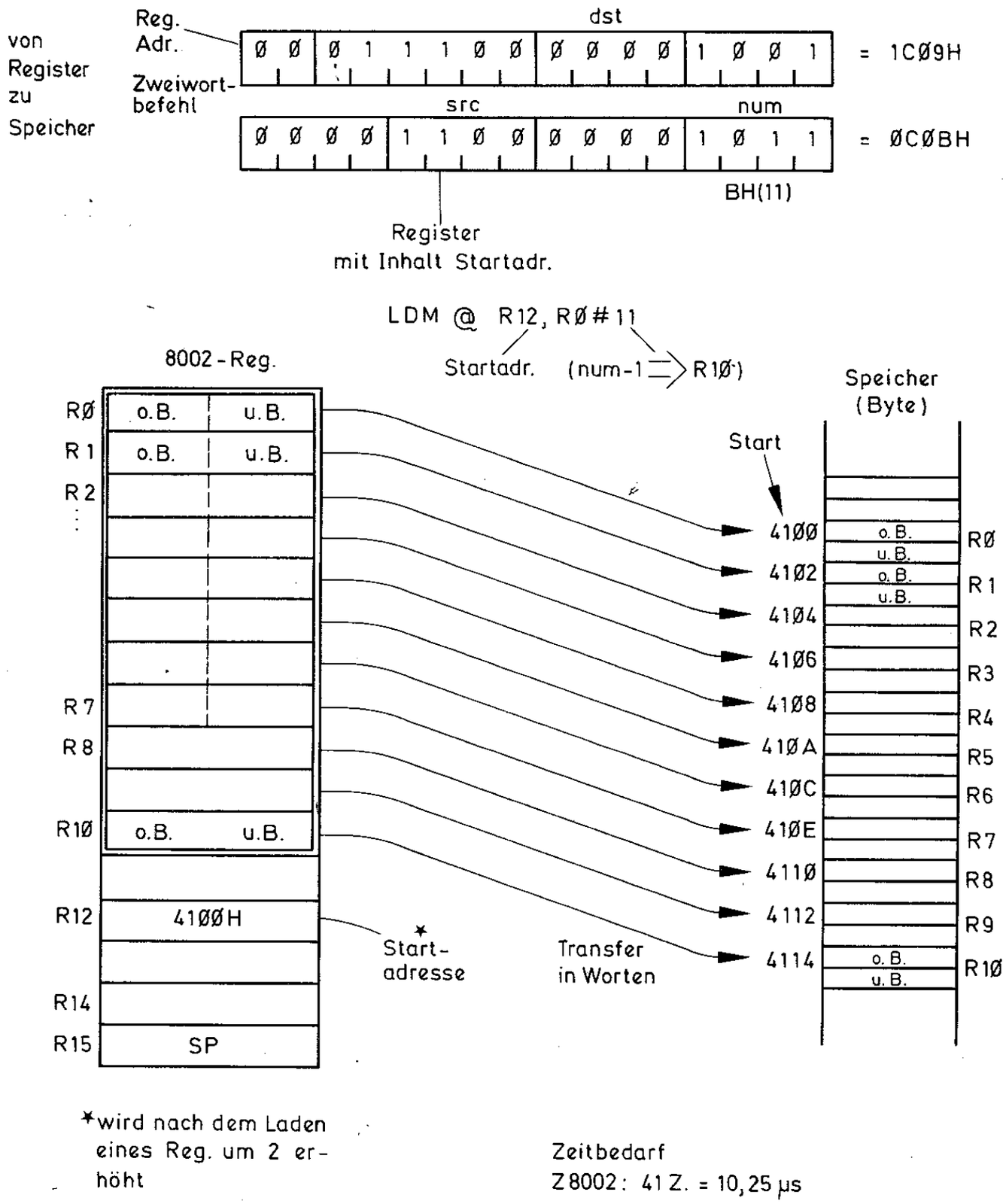


Bild 6.22

(Stapel um 2 Stellen dekrementiert zum Ablegen des alten PC). Er lautet BSR (Branch to Subroutine) für den 6800 und den 6809. Für 16-bit-CPUs lautet der Befehl ebenso CALL für den 8086 und BSR für den 68000 (gleich wie für 6800 und 6809).

Für alle gleichlautenden und gleichbedeutenden Befehle gilt dies: Bei CALL wird der alte PC-Stand gerettet, d.h. der Stand des auf den nächstfolgenden Befehl nach CALL. Dieser PC wird auf den Stapel gelegt (sprich wörtlich oben auf). Für den Z 8000 gilt dies genauso. **Bild 6.23** zeigt die Funktion des CALL-Befehls für den Z 8000 in der indirekten

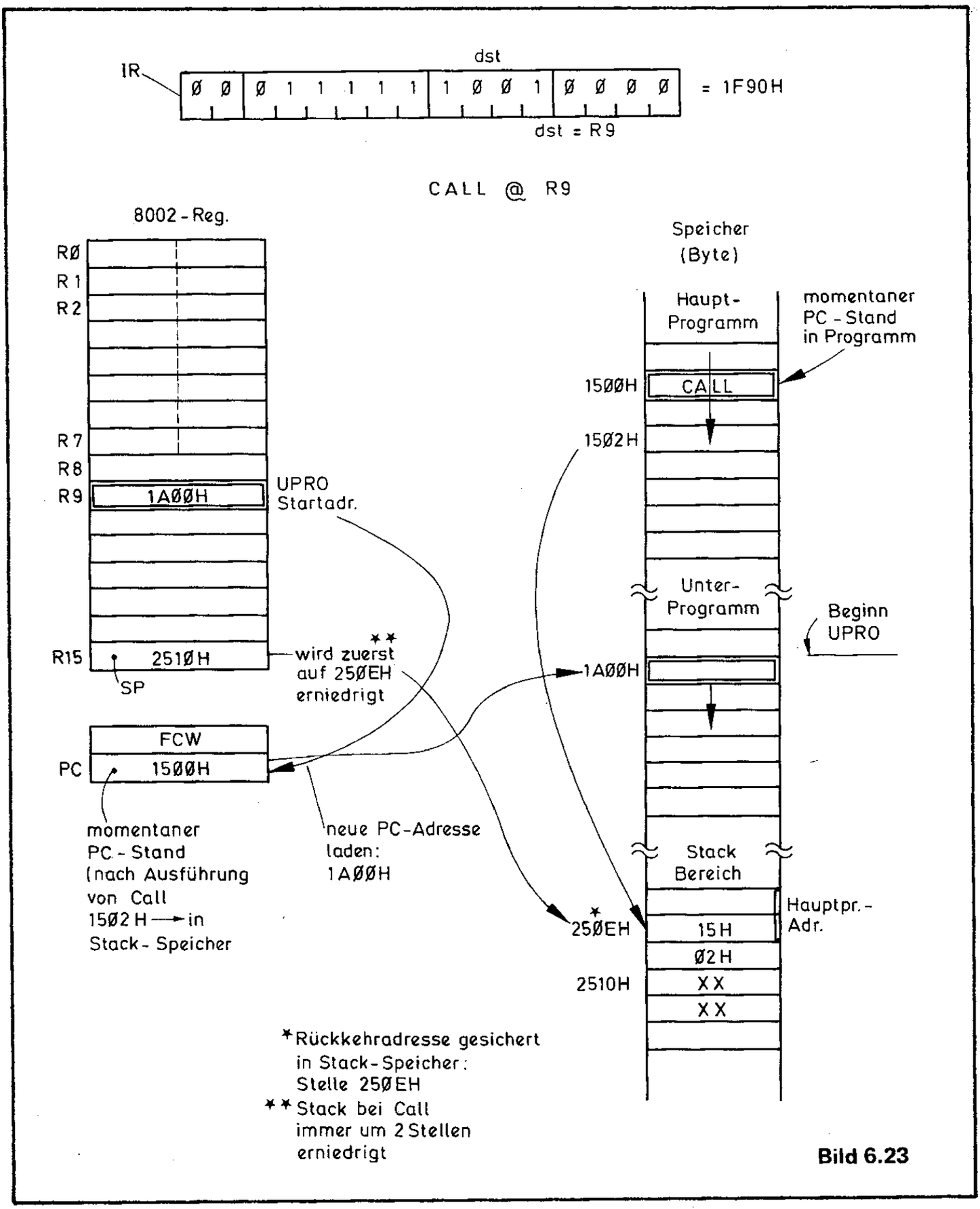


Bild 6.23

Adressierungsart. Für CALL ist überdies die indizierte und direkte Adressierungsart anwendbar.

CALL @R9

benützt den Inhalt des Registers R 9 als Startadresse für das Unterprogramm (UPRO). Der PC-Stand nach der Ausführung von CALL ist mit % 1502 angegeben. Die Rückkehradresse ist somit dieser Wert. Er wird auf den Stapel gelegt, wobei % 2510 des Stapelzeigers auf % 250E reduziert wird. Der UPRO-Start beginnt bei % 1A00. Die indirekte Adressierungsart ist für den CALL-Befehl des Z 8000 die schnellste. Sie ist mit 10 Zy-

klen für den nichtsegmentierten Modus angegeben. Im Gegensatz zur üblichen Form der indirekten Adressierung wird mit der Registeradresse kein Speicherplatz adressiert, sondern ein interner Transfer von Register zu PC ausgeführt. Bei der direkten Adressierung steht die Anfangsadresse des UPROs unmittelbar nach dem Befehlswort im Codespeicher.

Betrachtet man nun die segmentierte Form des CALL-Befehls, steht in Analogie zu Bild 6.23 ein Doppelwort-Register als Quelle für die UPRO-Startadresse. Der Befehl würde lauten (nur für Z 8001):

CALL @RR 8

In R 8 steht in diesem Fall die Segmentadresse, wobei das MSB des Registerwerts auf 1 stehen muß (langer Offset). In R 9 steht die UPRO-Startadresse in dem Segment, wie schon voran. Für den Stapelzeiger R 15 gilt vor der Adressierung des Stackbereichs die Reduzierung um 4 Stellen statt 2 im nichtsegmentierten Modus. Zuerst wird der PC-Offset und anschließend der PC-Segmentwert auf den Stapel gelegt. Segmentwert und Offset steht jeweils in einem Register. Ob nun das UPRO im gleichen Segment liegt, von dem der Sprung erfolgt, oder in einem anderen, ist gleichgültig. Ist die Z 8001-CPU im segmentierten Modus, wird mit

CALL @RR 2	indirekt (geradzahliger Wert für RR bis 12)
CALL ≪27≫% 4100	direkt (langer Offset)
CALL ≪27≫% 4100 (R 5)	indiziert

ein legaler Segmentübertritt eingeleitet.

Ein Rücksprung aus dem UPRO wird mit RET (Return from Procedure) ausgeführt. **Bild 6.24** zeigt in Analogie zu CALL in Bild 6.23 mit den gleichen Adreßwerten den Rücksprung in das Hauptprogramm an der Stelle % 1502. Der Stapelzeiger adressiert den Stapelspeicher mit der noch in ihm stehenden Adresse % 250E die Hauptprogramm-Rücksprungadresse und lädt sie in den PC. Es gilt hierzu noch die Verfahrensweise, daß immer in das Programm zurückgesprungen wird, in dem dieses UPRO aufgerufen wurde. Es spielt hier keine Rolle, ob es ein höher priorisiertes UPRO war oder tatsächlich das Hauptprogramm. Die Rückkehr aus Unter-UPROs geschieht stufenweise. Dies dürfte als bekannt vorausgesetzt sein. Der RET-Befehl enthält das Bedingungsfeld "cc". In Abschnitt 3.3 ist zu entnehmen, welcher Code für den jeweiligen Entscheidungsfall einzusetzen ist. Setzt man 1000 ein (steht für "immer erfüllt"), benötigt der Assembler keine weitere Angabe als nur RET.

Im segmentierten Modus verhält sich der Ablauf bezüglich des Stapels umgekehrt. Der Stapelzeiger enthält noch die Adresse des Segmentwerts des vorhergehenden Programms, der auf den Stapel gelegt wurde. Dieser Segmentwert wird zuerst in die CPU zurückgeholt. Nach dem Inkrementieren um 2 Stellen folgt der Offsetwert des vorhergehenden Programms. Nach diesem Holvorgang ist der PC-Segment- und PC-Offsetteil als neuer Programmstatus gesetzt und der Stapelzeiger ist um 4 Stellen höher als vor der RET-Operation. Das folgende Beispiel sollte zeigen, wie mit dem Zilog-Assembler eine Programmbearbeitung über mehrere Segmente gehandhabt wird. Man setzt mit Labels, wie sie schon in Abschnitt 3.3 und an anderen Stellen erwähnt wurden. Für die Labels stehen direkte Adressen, woraus sich für die Anwendung von CALL die direkte Adressierungsart ergibt. Der Assembler setzt die richtige Adresse für das Label automatisch ein. Es erfordert nicht einmal, zu Beginn die effektive Adresse zu definieren, weil das durch die Bezeichnung des Programms mit Name MODULE vordefiniert wird. Die genaue Definition geschieht durch den Linker. Der Linker bindet die einzelnen Objektdateien zu einem zusammenhängenden Block. Er ordnet jedem Modulname seine spezifische Startadresse automatisch zu, da die Längen für ihn genau berechenbar (mit END Name) sind. Der Anwender braucht sich hierum nicht zu kümmern. Die allgemein gültige

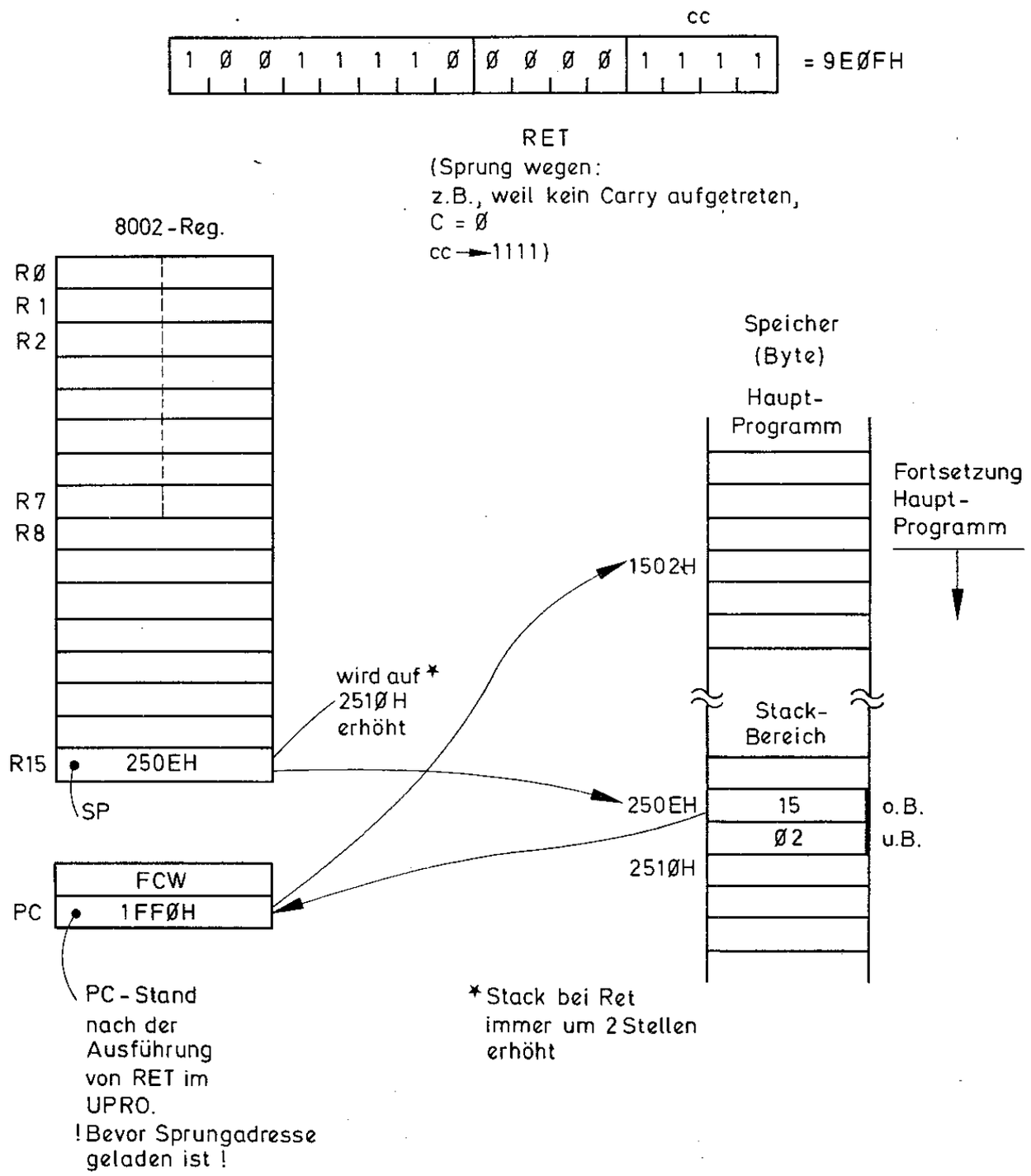


Bild 6.24

Formulierung von Labels bleibt dem Anwender überlassen. An dieser Stelle wird hierfür "Level_1, Level_2, Level_3" eingesetzt und mit "Level_2_A" der erste Durchlauf bezeichnet. Den Verlauf der Prozedur zeigt prinzipiell das **Bild 6.25.**

Adresse	**Opcode**	**Zeile**	
		1	Feldvergleich MODULE
		2	
		3	$ SEGMENTED
		4	
		5	INTERNAL

```
Adresse  Opcode          Zeile
                          6
                          7   $ SECTION MAIN
                          8
                          9   Level_1 PROCEDURE
                         10   ENTRY
0000     5F00 5100       11   CALL  Level_2_A
0000     0000
                          .
                          .
                          .
00F0     5F00 5100       30   CALL  Level_2_B          ! zweiter Durchlauf !
00F4     000E
                         31   HALT
                         32   END Level_1
                         33
                         34   $ SECTION SUBROUTINES
                         35
0000                     36   Level_2_A PROCEDURE
                         37   ENTRY
0000     5F00 5100       40   CALL  Level_2_B
0004     000E
0006     5F00 5100       41   CALL  Level_3_A
000A     0000
000C     9E08            42   RET
000E                     43   END Level_2_A
                         44
000E                     45   Level_2_B PROCEDURE
                         46   ENTRY
000E     5F00 5100       47   CALL  Level_3_A
0012     0000
0014     9E08            48   RET
0016                     49   END  Level_2_B
                         50
                         51   $ SECTION UTILITY
                         52
0000                     53   Level_3 PROCEDURE
                         54   ENTRY
0000     9E08            55   RET
0002                     56   END  Level_3
                         57
                         58   END  Feldvergleich
```

Wie man hier feststellen kann, besteht der beiden Unterprogrammbereiche die UPRO-Prozedur nur in dem Aufruf Level_2_B und Level_3_A und dem

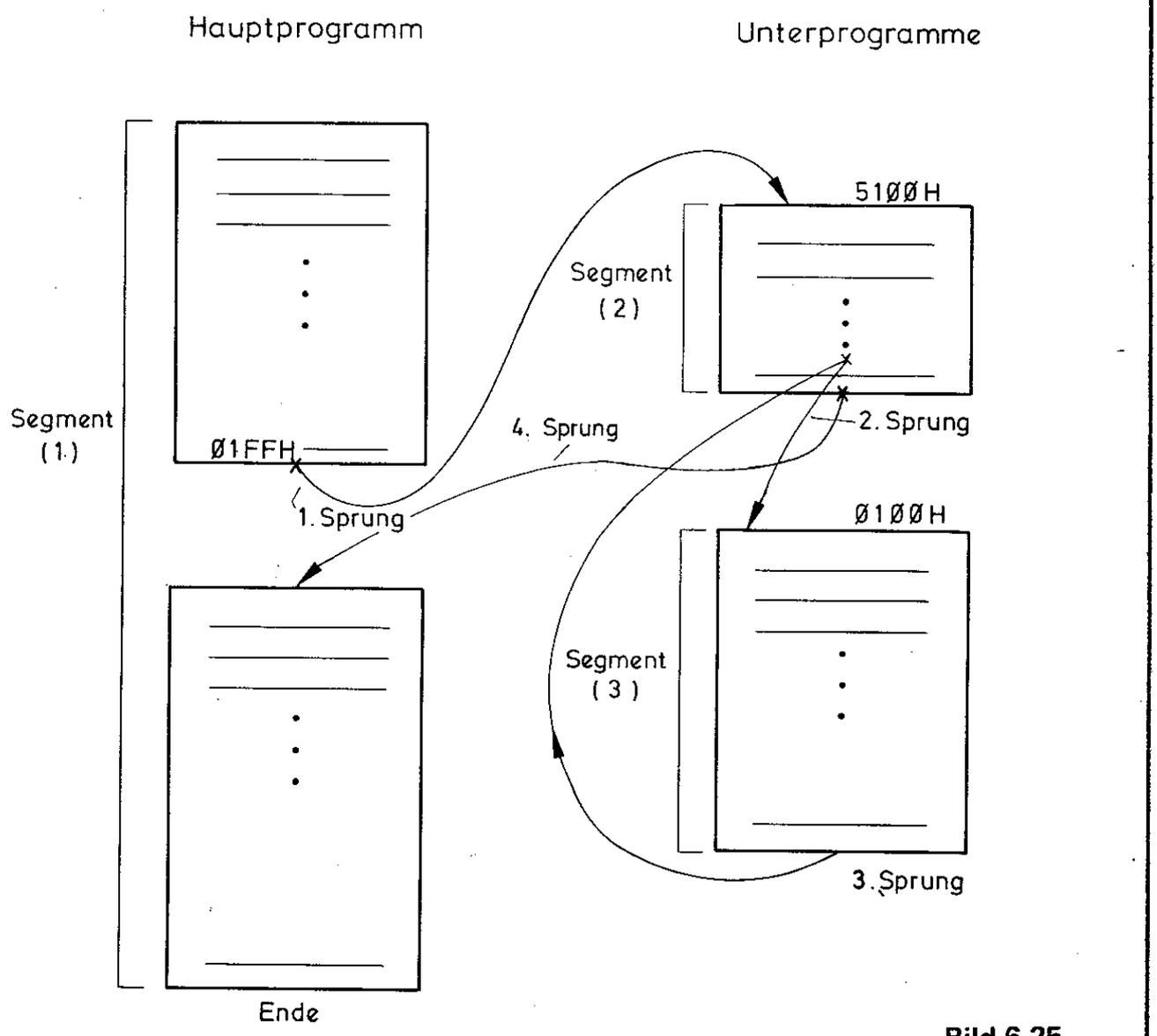


Bild 6.25

Rücksprung-Befehl. Alles andere ist bereits vorgegeben und im Katalog der "Subroutines" enthalten. Man merkt dabei eigentlich garnichts von Segmentsprüngen. Sie werden von dem Z 8000-System einfach ausgeführt, ohne daß eine Ankündigung besonderer Art erfolgt, würde man dies nicht ausdrücklich wünschen. Der Wert % 5F00 für den Opcode von CALL ist leicht erklärbar. In dem in Bild 6.23 gezeigten Codefeld wird für die direkte Adressierung der Wert 01 in den ersten 2 Feldern eingesetzt. Damit ist 5F klar. Im dst-Feld wird 0 eingesetzt. Damit ergibt sich % 5F00. Für RET orientiere man sich im Bild 6.24 (cc = 1000).

Ein unmittelbarer Segmentübertritt bei der Z 8001-CPU (SEG = 1) erfolgt nach einem Trap oder einem Interrupt. Der Systemstapelzeiger RR 14 schiebt in der Reihenfolge den PC-Offset, den PC-Segmentwert, den FCW-Inhalt und zuletzt die Kennung in den Systemstapel. Der Systemstapelzeiger wurde im Verlauf der Interrupt-/Trapanerkennung um 8 Stellen erniedrigt (4 Worte). Die **IRET-Operation** behandelt die Rückkehr aus einer Interrupt- oder Trap-Servicerouti-ne. Zuerst wird das Kennungswort vom Stapel ausgelesen. Dieses Wort war der auslösende Grund für den CPU-Trap. Jetzt wird es nicht mehr gebraucht und daher von der CPU nicht mehr übernommen. Wurde vor IRET mit der Kennung nichts mehr angestellt, geht die Kennung praktisch verloren. Der nächste, durch den Stapelzeiger adressierte Wert ist der alte FCW-Wert. Dieser wird direkt in das FCW-Register transferiert. Zuletzt wird der PC-Segmentwert und

der PC-Offset in den PC geholt. Die CPU kann damit an dieser Stelle ihre Arbeit fortsetzen, die mit diesem Status definiert ist. Dieser Segmentübertritt ist noch unauffälliger wie vorher mit CALL. Allgemein läßt sich für die Z 8000-CPU feststellen, daß ein Segmentübertritt ausgesprochen problemlos abläuft.

Der letzte Fall eines Segmentsprungs im Rahmen dieser Zugehörigkeit betrifft den elementaren JP-Befehl. Er bezeichnet den einfachsten Fall und wird auch sehr häufig angewandt. **Bild 6.26** zeigt diesen Fall grafisch für die nichtsegmentierte Version. Der Befehl ist als Branch-Funktion aufzufassen und beinhaltet hierfür das 4-bit-cc-Feld. Die verschiedenen Bedingungscodes sind aus der Tabelle in Abschnitt 3.3 zu entnehmen. Das gezeigte Beispiel betrifft einen bedingten Sprung (Sprung, wenn C = 1 erfüllt ist). Der Befehl ändert keine Registerinhalte und keine Flags. Es wird ausschließlich der Programmzähler betroffen. Der Wert % 175C wird aus dem Programmcodespeicher geholt und in den PC übertragen. Wie für CALL, so sind auch für JP die Adressierungsarten indirekt, direkt und indiziert anwendbar. Im segmentierten Modus werden die drei Varianten folgendermaßen geschrieben:

```
JP    @RR 2                    indirekt
(geradzahliger Wert für RR 2 bis 12)

JP  ≪ 100 ≫ % 5100             direkt
(langer Offset)

JP  ≪ 100 ≫ % 5100(R9)         indiziert
```

Die gezeigte Form betrifft einen Sprung ohne Bedingung (cc = 1000). Für den Zilog-Assembler wird im allgemeinen keine Angabe einer absoluten Adresse gemacht, wie schon mehrfach erwähnt. Die örtliche Spezifikation mit Hilfe eines Labels ist auch für JP die Regel.

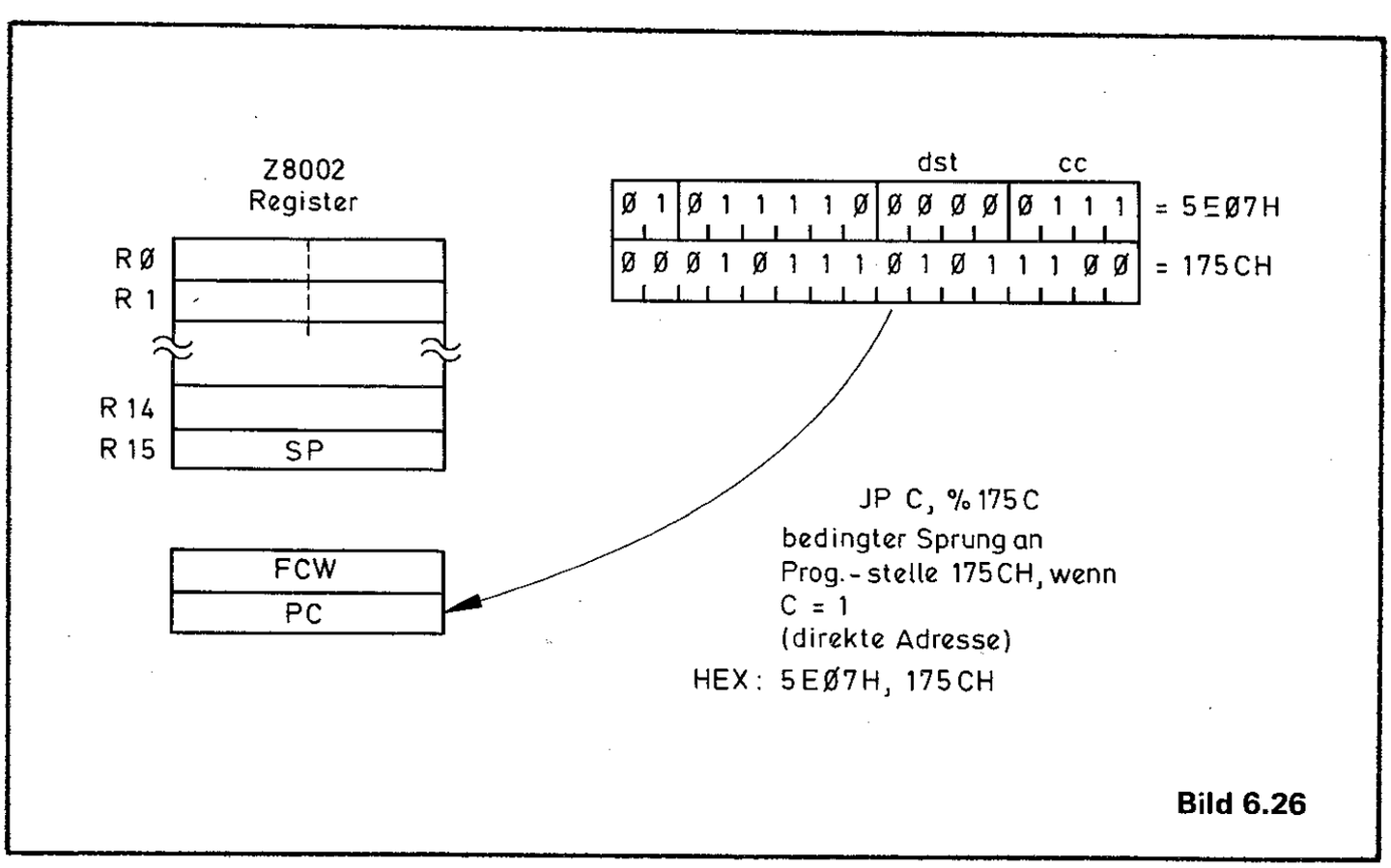


Bild 6.26

6.2. VERIFIKATION EINZELNER OPERATIONEN

Assemblerprogramme werden auf zwei Ziele hin entwickelt. Entweder ist man an minimaler Ausführungszeit interessiert oder die logische Gestaltung des Programms soll sehr übersichtlich sein. Im ersten Fall ist im Rahmen der Software jedes Mittel recht, um eine zeitoptimale Ausführung zu erzielen. Es treten dann teilweise merkwürdige Befehlsfolgen auf, die aber letztlich der Zeitoptimierung zuzuschreiben sind. Es gibt Aufgaben, die sich auf verschiedene Wege lösen lassen. Sie unterscheiden sich zumeist in ihrer Ausführungszeit. Die Gesamtzahl der erforderlichen Zyklen ist unterschiedlich. Hier ist die eine oder andere Mikrosekunde zu sparen. Bei Benchmark-Untersuchungen ist dies das angestrebte Ziel.

Im zweiten Fall geht es um die Übersichtlichkeit eines Assemblerprogramms. Man spricht von einer codeoptimierten Ausführung des Programms. Es werden im Assembler-Quellprogramm genau die Befehle verwendet, die einen am nächstliegenden Zusammenhang zur Aufgabe bilden. Wird der Aufbau des Programms an Hand des Flußdiagramms noch hinreichend strukturiert, d.h. mit weniger Schleifen und einem Sprung über nur einen Block versehen, ist das Softwareproblem für beliebige Anwender leichter überschaubar (sofern überhaupt ein Verständnis vieler Anwender erwünscht ist). Daß ein in diese Richtung entwickeltes Programm nicht zeitoptimal sein kann, ist leicht einsehbar. Welchen Fall man hier den Vorzug gibt, hängt von den Wünschen des Anwenders und dem Einsatz des Programms ab. Generell muß man bemerken, daß ein so schneller Prozessor wie der Z 8000 der Rücksichtnahme einer besseren Verständlichkeit einer Softwareentwicklung entgegenkommt. Er ist vielfach schnell genug, um gewisse Umwege bei der Programmentwicklung auf Kosten der Zeit auszugleichen. Für den Z 8000 gilt vielmehr die Erfahrung, daß für eine Reihe von Miniprozeduren die Ausführung sowohl zeit- als auch codeoptimal ist. Das trifft man selten. Auf die Fälle, in denen im Einzelnen noch Zeitvorteile zu erreichen sind, soll hier eingegangen werden.

Ein Beispiel zeigt auf der Befehlsebene z.B. der Ausdruck

```
TEST   R 0
```

Der Inhalt von R 0 wird an jeder Stelle logisch ODER verknüpft. Es ändern sich demgemäß die Flags Z, S und (eingeschränkt) P. Der eigentliche Operand im Register bleibt unverändert. Diese Operation erfordert im nichtsegmentierten Modus und in der Registeradressierung **7 Zyklen.** Legt man es auf einen Test über die Flags an, kann für diesen Befehl in Analogie

```
AND   R 0, R 0
```

verwendet werden. Die logische UND-Verknüpfung, die dieser Befehl mit dem Operand in R0 bewirkt, hat letztlich dasselbe Ziel wie TEST R0. Wendet man AND an, verknüpft z.B. % F4 mit F4, bleibt derselbe Wert in R0 zurück. Die Ausführung von AND bewirkt jedoch das Ergebnis in **nur 4 Zyklen** (unter gleichen Voraussetzungen).

Ein weiterer Fall gilt geradezu als klassisch. Will man einen Registerinhalt eliminieren (Null setzen), gibt es drei Möglichkeiten mit drei unterschiedlichen Zeiten. Würde man verständlicherweise den codeoptimalen Befehl CLR (Null setzen) verwenden, ist man zeitlich nicht so gut beraten. Die Auflistung

```
CLR  R 3        (7 Zyklen)
LDK  R 3, L 0   (5 Zyklen)
SUB  R 3, R 3   (4 Zyklen)
```

zeigt, daß der SUB-Befehl mit der Registeradressierung am wenigsten Zeit erfordert.

Eine Verifikation betrifft auch einen Befehl selbst. Wie aus der Befehlsliste in Kapitel 3 ersichtlich ist, existiert für den AND-Befehl keine 32-bit-Ausführung. Hierfür gibt es jedoch auch eine Abhilfe.

Man läßt die CPU

```
AND   R0 , R2
AND   R1 , R3
```

ausführen und erzielt damit die UND-Verknüpfung in R0 und R1 bzw. RR0. Rechnet man den Zeitbedarf zusammen, erfordert dies 8 Zyklen. Wollte man hierfür wieder ein Äquivalent eines Befehls einsetzen, kann man das tun. Man setzt dafür

```
TESTL   RR 0
```

Dieser Befehl ist für den 32-bit-Operand in RR 0 gedacht. Die Ausführung ist der Art, wie bereits beschrieben. Der Befehl benötigt jedoch 13 Zyklen und ist damit länger als die vorher genannten beiden AND-Befehle zusammen.

Der Z 8000 ist ein optimierter Prozessor. Wollte man beispielsweise die Stringbefehle "laden, vergleichen, übersetzen mit Test" in einem expliziten Programm nachbilden, wird man in jedem Fall mehr Zeit benötigen, als der jeweils spezifizierte Befehl erfordert. Hier gibt es nichts zu holen. Man könnte es beispielsweise bei dem Multiplikationsbefehl versuchen, ob hier noch ein paar Zyklen einzusparen sind.

```
MULT   RR0, R3
```

erfordert 70 Zyklen. Man nehme an, folgende beliebige Werte wären zu multiplizieren:

```
R 0  =  % 4F36
R 3  =  % 2510
```

Spaltet man den Wert von R3 (Multiplikator) in die nächst kleinere Zweierpotenz auf und gibt den Rest zu 2510H als Summe an, ergibt sich daraus

$$2^{13} = 8192$$

Verschiebung von R0 um 13 Stellen nach links entspricht einer Multiplikation mit 8192.

2510H = 9488; das Produkt von R0 mit R3 lautet

```
R0 (8192 + 1296)
```

Die nächst kleinere Zweierpotenz von 1296 ist $2^{10} = 1024$. Folglich ist

```
R0 (8192 + 1024 + 256 + 16)
```

komplett als Summenprodukt mit Zweierpotenzen aufgesplittet. Man verschiebt den Multiplikand in R0 erst um 4 Stellen, dann wieder um 4, dann um 2 und nochmal zuletzt um 3 Stellen und addiert jeweils die Zwischenwerte. Ein Programm der Multiplikation der gegebenen Operanden ohne den MULT-Befehl sieht folgendermaßen aus:

Zeile	Befehl	Bezeichnung	Zyklen
.			
.			
.			
10	SUBL	RR 0 , RR 0	[8]
11	LDL	RR 4 , #% 00004F36	[12]
12	SLLL	RR 4 , #4	[25]
13	ADDL	RR 0 , RR 4	[8]
14	SLLL	RR 4 , #4	[25]
15	ADDL	RR 0 , RR 4	[8]
16	SLLL	RR 4 , #2	[19]
17	ADDL	RR 0 , RR 4	[8]
18	SLLL	RR 4 , #3	[22]
19	ADDL	RR 0 , RR 4	[8]
20		END Multiplikation	

Nach diesem Miniprogramm steht der gleiche Wert in RR 0 wie durch

```
MULT   RR 0 , R 3
```

Der Erfolg bei dieser Umgehung des MULT-Befehls ist jedoch auch gleich Null, weil diese Methode sehr aufwendig ist. Der Zeitbedarf ist mit 143 Zyklen gerade gut doppelt so hoch als die Ausführung der Multiplikation von R0 · R3 mit dem MULT-Befehl. Von Gewinn kann also keine Rede sein. Die Z 8000-Befehle sind in sich bereits optimiert; das ist ein Faktum. Effektive Gewinne sind nur in einer geschickten Programmierung zu erzielen, der überlicherweise eine sehr präzise Problemanalyse vorausgeht. Um so besser man das zu erfassende Problem verstanden hat, um so leichter kann ein zeitoptimales Programm entworfen werden.

6.3 EIN DENKSPIEL PROGRAMMIERT, DAS SOLITAIRE-SPIEL

Ein Computer ist mitunter auch ein Spielzeug; Spielzeug nicht im Vergleich zu anderen Computern, wie dies in Kapitel 1 gemeint war, sondern Spielzeug im Sinn der Entspannung, des Vergnügens, der Anregung des Geistes.
. . . In diesem Abschnitt wird ein Spiel vorgestellt, das eine gezielte Operation zum Ziel hat. Es ist kein Trivialspiel wie Star-Treck oder Master Mind. Die Auswahl des Spiels wurde in einem ursächlichen Zusammenhang mit dem Z 8000-Befehlssatz getroffen. Genaugenommen zielt es auf die Anwendung eines bestimmten Befehls ab. Es sei hier im Voraus bemerkt, daß das betreffende Spiel mit jedem kleinen (üblichen) 8-bit-Prozessor programmierbar ist. Für den 6502 gibt es ein Assemblerprogramm für dieses Spiel. Ganz besonders einfach ist die Programmierung mit einem Basic-Computer wie den cbm 2001, den TRS 80, den MZ 80K und ähnlichen. Es ist also ein ernsthafter Irrtum, etwa zu glauben, es ist die Größenordnung eines Z 8000-Systems erforderlich, um das besagte Spiel zu programmieren. Es geht im folgenden um eine einfache, prinzipielle Anwendung eines Z 8000-Stringbefehls. Die Anwendung dieses Befehls bringt tatsächlich eine effektive Vereinfachung bei der Programmierung in Assembler gegenüber anderen Prozessoren.

Das Spiel nennt sich Solitaire- oder Einsiedlerspiel und dürfte sicherlich denjenigen Lesern bekannt sein, die Freunde kniffliger Denkspiele sind. **Bild 6.27** zeigt das einfache Spielfeld in Form eines Punktrasters. Wie auch immer die Ausführung des Spielfeldes ausgebildet sein mag, alle bezeichneten Punkte des Rasters bestehen aus Löcher, in denen eine Spielfigur eingesetzt ist. Die Spielfiguren sind entweder Kugeln oder Doppelkegel. Das Bild zeigt die Ausgangsstellung. Das Mittelpunktloch bleibt frei. Die schwarzen Punkte sind mit Spielfiguren belegt. Der Spieler darf nur mit einer Spielfigur eine Nachbarfigur überspringen und in ein darauffolgendes, freies Loch einspringen. **Bild 6.28** zeigt einen erlaubten Sprung zu Beginn. Die übersprungene Spielfigur muß darauf aus dem Spielfeld entfernt werden. Es ist auch erlaubt, mit einer Figur mehrere solcher Sprünge auszuführen (bei fortgeschrittenem Spiel), wenn die Zwischenräume dies zulassen. Der Verlauf bzw. die Regeln sind sehr einfach. Das Ziel des Spiels ist ebenso einfach. Nach dem letzten Sprung soll sich die einzig verbleibende Spielfigur im Mittelpunktloch befinden und alle anderen Löcher unbelegt sein, wie dies **Bild 6.29** zeigt. Das sieht alles sehr einfach aus. Man wird jedoch sehr schnell eines besseren belehrt werden, wenn man sich an dem Herausfinden der richtigen Reihenfolge der Sprünge versucht. Sehr schnell können hier Stunden verrinnen, wenn jemand spontan die richtige Reihenfolge herausfinden will. So einfach ist es ganz bestimmt nicht. Eine Lösung wird im Verlauf nicht angegeben weil das nicht zum Thema gehört. Es wird nur in **Bild 6.30** die Stellung einer richtigen Reihenfolge angegeben, die nach dem 17. Sprung erreicht wurde. Es gibt drei bisher bekannte Reihenfolgen, die zum Ziel in Bild 6.29 führen. Von einer dieser Reihenfolgen ist der gezeigte ein markanter Zwischenstand.

Das Prinzip des Spiels ist somit hinreichend dargestellt, so daß auf die Programmierung eingegangen werden kann. Für eine Softwarelösung dieser Aufgabe besteht primär das Erfordernis einer genauen Lokalisierung jedes einzelnen Spielpunktes. Das gilt ebenso für die Ränder des Spielfeldes, d.h. die Definition von nicht erlaubten Spielfeldpunkten oder virtuellen Punkten. Die einfachste Lösung hierfür ist ein Koordinatenfeld. Man legt hierzu ein Koordinatenkreuz fest, in dem das Spielfeld im ersten Quadranten liegt. Die Begrenzung

Ausgangsstellung

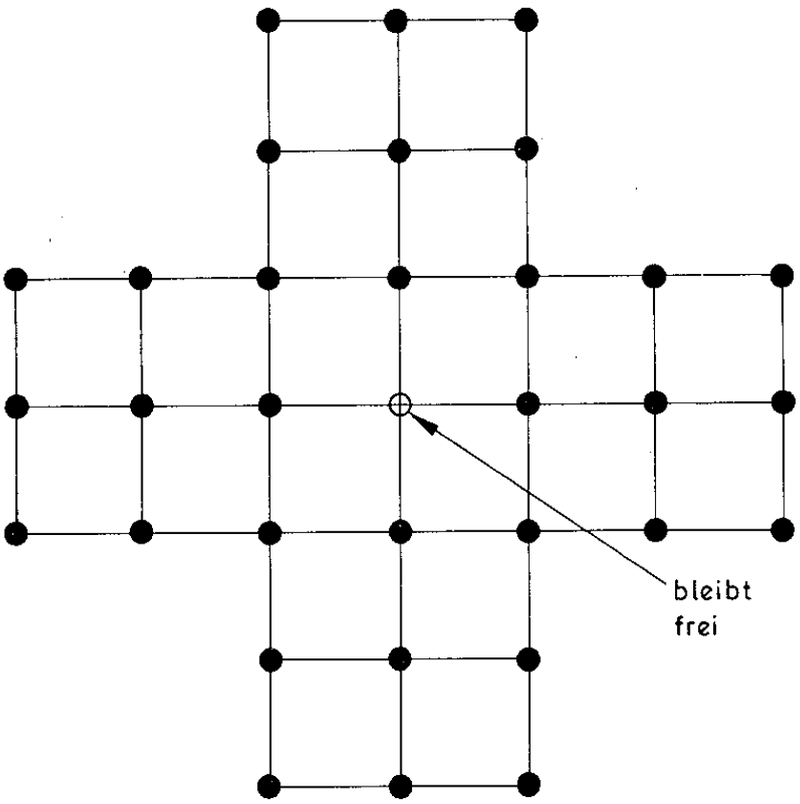


Bild 6.27

Stellung nach
erstem Sprung

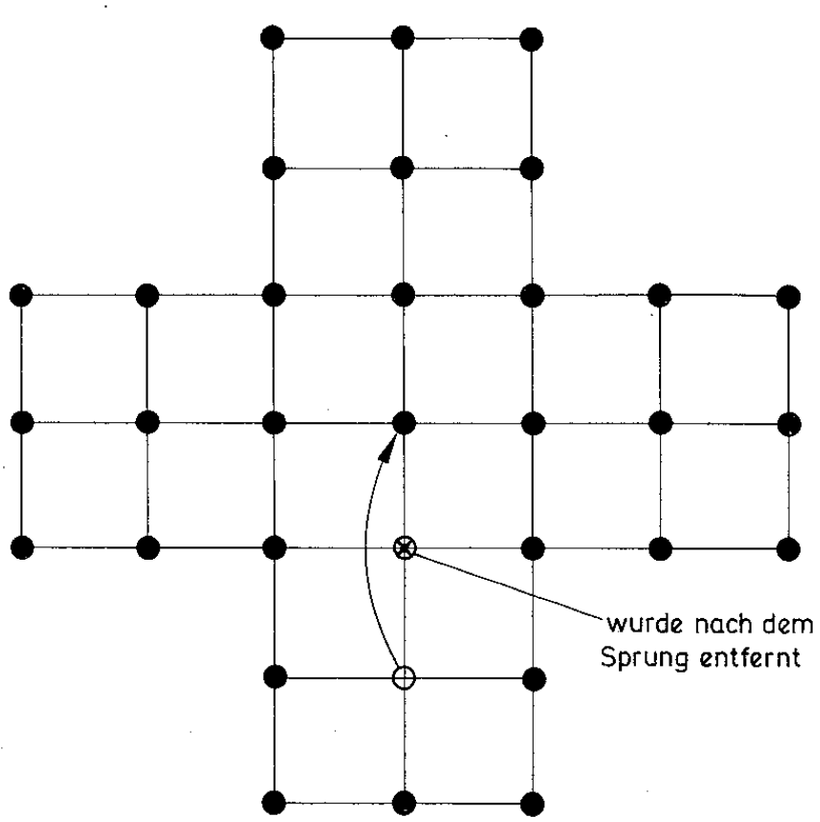


Bild 6.28

Endstellung

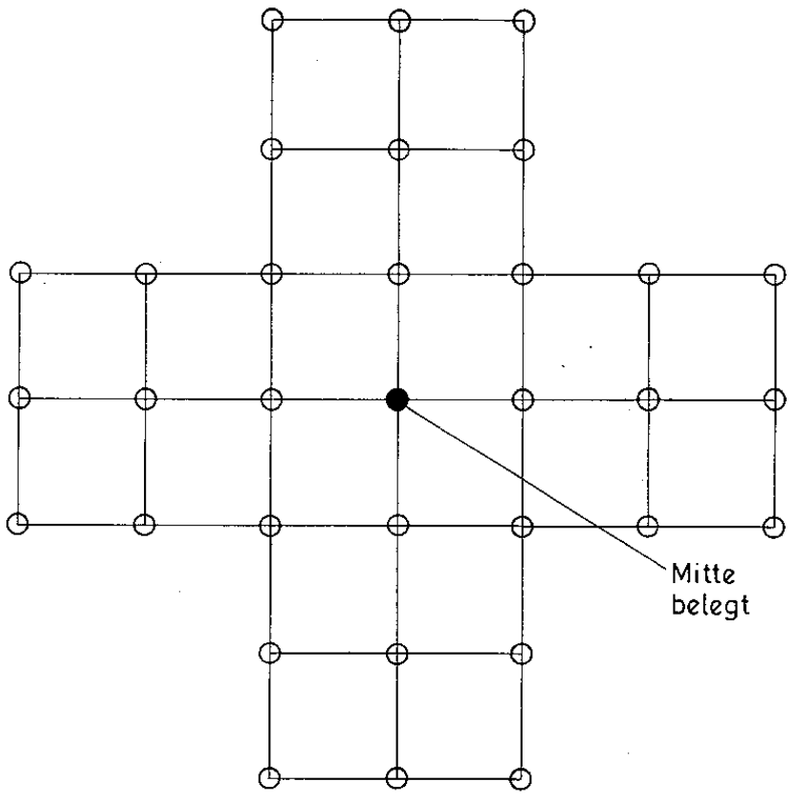
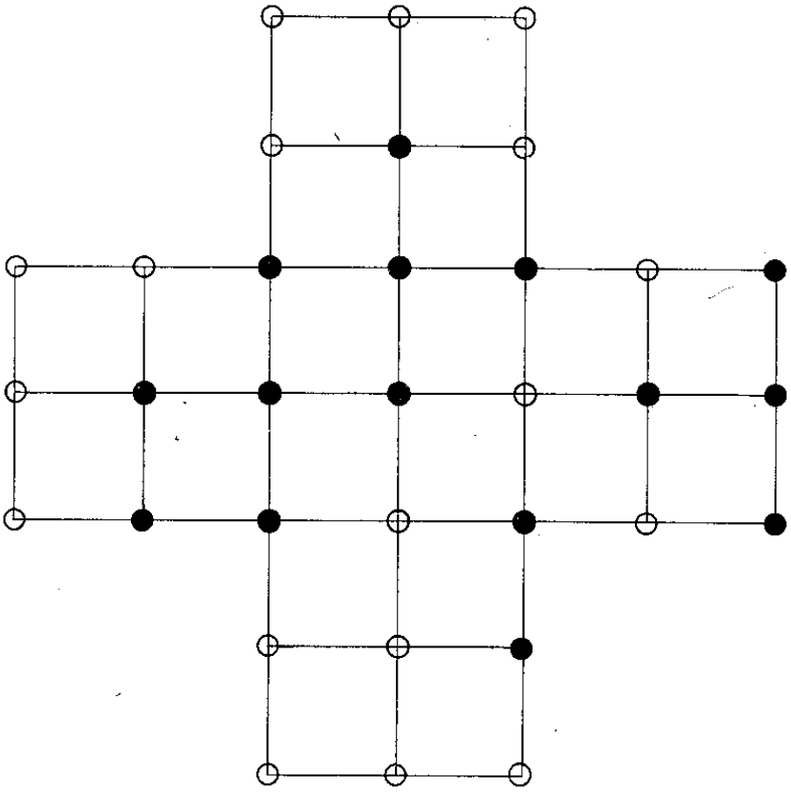


Bild 6.29

Stellung nach
dem 17. Sprung



● belegt

○ freie Stelle

Bild 6.30

ist so zu legen, daß der linke und obere Spielfeldrand durch die Ordinate bzw. Abszisse begrenzt wird. Man kann auf diese Weise negative Koordinatenpunkte als unerlaubte Spielfeldpunkte bezeichnen. Das Koordinatenfeld mit dem ersten Quadranten zeigt **Bild 6.31.** Die erlaubten Spielfeldpunkte sind die Kreise, die durch eine umgrenzende, gestrichelte Linie zusammengefaßt sind. Die mit Kreuzen bezeichneten Koordinatenpunkte in dem 6 x 6 Punktfeld sind bereits illegale Spielfeldpunkte.

Es besteht in der Tat die ernsthafte Frage, ob eine genaue Eingrenzung des Spielfeldes notwendig ist. Im Sinn einer exakten Einhaltung der Regeln ist dies jedoch empfehlenswert, denn es wäre fast lächerlich, würde sich der Computer durch fehlende Definitionen aus dem "Konzept" bringen lassen. Grundsätzlich ist zu bemerken, daß der Computer nur als Mittel zum Zweck dient. Der Computer erlaubt die Ausführung des Spiels über die Tastatur und, bei entsprechenden Erweiterungen, die Kontrolle auf einem Bildschirm. Er ist nicht als Spielpartner zu betrachten, wie dies bei Schachprogrammen der Fall ist. Die Softwarelösung für das Spiel soll alle Regeln beinhalten, die im Rahmen des Spiels einzuhalten sind. Versucht ein Spieler außerhalb des Feldes zu springen, sei es aus Unwissen oder als Überlistungsversuch gedacht, muß der Computer ihm anzeigen, daß er einen illegalen Sprung ausgeführt hat. Dazu ist softwaremäßig zu definieren, welche Sprünge illegal sind. Bild 6.31 zeigt den Ansatz eines legalen Sprungs in horizontaler Richtung (x-Richtung, Sprung von P1 (1/2) nach P2 (3/2)). Der Sprung ist dann erlaubt, wenn Punkt P (2/2) mit einer Spielfigur belegt ist. Diagonale Sprünge sind nicht erlaubt. Die folgende Auflistung der Bedingungen (auch Regeln) definiert die Richtlinie für die Entwicklung des Solitaire-Programms:

- Definitionsfeld legaler Spielpunkte in einem 6 x 6 Raster, abzüglich 16 illegale Randpunkte
- 16 illegale Randpunkte mit den Koordinaten:
  $P_X1 \ldots 4$ (0/0; 0/1; 1/0; 1/1),
  $P_X5 \ldots 8$ (0/5; 0/6; 1/5; 1/6)
  $P_X9 \ldots 12$ (5/0; 6/0; 5/1; 6/1),
  $P_X13 \ldots 16$ (5/5; 6/5; 5/6; 6/6)
- Illegale Randpunkte außerhalb des 6 x 6 Rasters:
  1. alle Punkte mit negativem Ordinaten- oder Abszissenwert
  2. alle Punkte mit einem Ordinaten- oder Abszissenwert, der größer als 6 ist.
- Sprungfelddefinition innerhalb legaler Spielfeldpunkte:
  Prüfung auf Horizontal- oder Vertikalsprung

$x_2 \neq x_1$	(bei $y_2 = y_1$)	horizontaler Sprung
$y_2 \neq y_1$	(bei $x_2 = x_1$)	vertikaler Sprung
$x_2 \neq x_1$	$y_2 \neq y_1$	illegaler Sprung

- Erste Sprungbedingung bei legalem Sprung in legalen Spielfeldpunkt

horizontal
$(y_2 = y_1)$: $x_1 > x_2 \rightarrow x_1 - 1 \rightarrow$ belegt
$x_1 < x_2 \rightarrow x_2 + 1 \rightarrow$ belegt

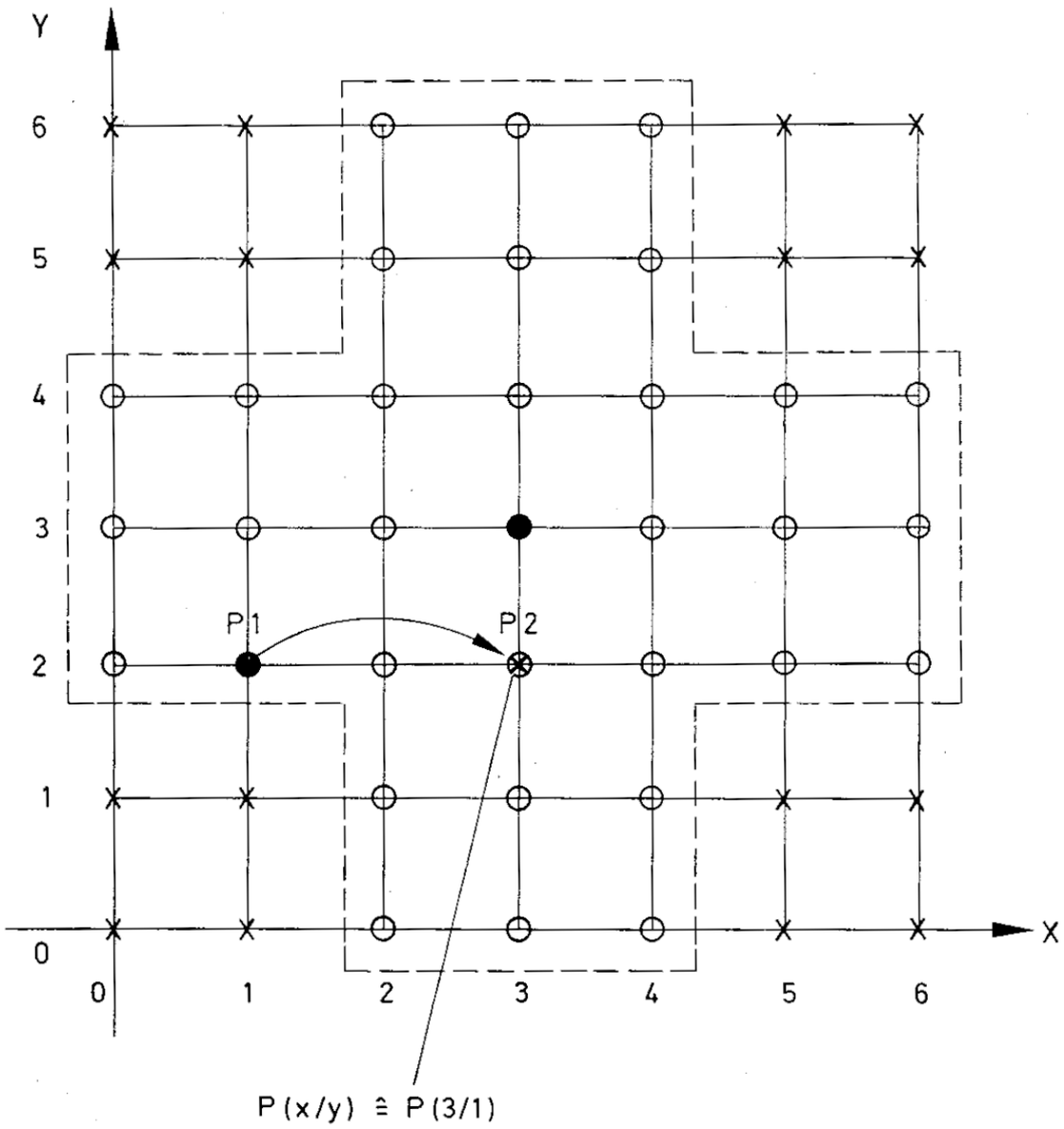
vertikal
$(x_2 = x_1)$: $y_1 > y_2 \rightarrow y_1 - 1 \rightarrow$ belegt
$y_1 < y_2 \rightarrow y_1 + 1 \rightarrow$ belegt

- Zweite Sprungbedingung bei legalem Sprung in legalen Spielfeldpunkt

horizontal
$(y_2 = y_1)$: $x_1 > x_2 \rightarrow x_1 - 2 \rightarrow$ frei
$x_1 < x_2 \rightarrow x_1 + 2 \rightarrow$ frei

vertikal
$(x_2 = x_1)$: $y_1 > y_2 \rightarrow y_1 - 2 \rightarrow$ frei
$y_1 < y_2 \rightarrow y_1 + 2 \rightarrow$ frei

Aus diesen zusammengefaßten Bedingungen ist ein Flußdiagramm entwickelbar, das eine sichere Bearbeitung eines Sprungs gewähren kann. Die vollständige softwaremäßige Behandlung ist in dem Flußdiagramm nach **Bild 6.32** integriert. Es behandelt die Prüfung aller Bedingungen, die für einen gültigen, legalen



P (x/y) ≙ P (3/1)

Sprung P1 zu P2 :

P1(1/2) ⟶ P2 (3/2)

nicht existente Spielplätze (X) :

0/0, 0/1, 1/0, 1/1 ;

5/0, 5/1, 6/0, 6/1 ; . . . usw.

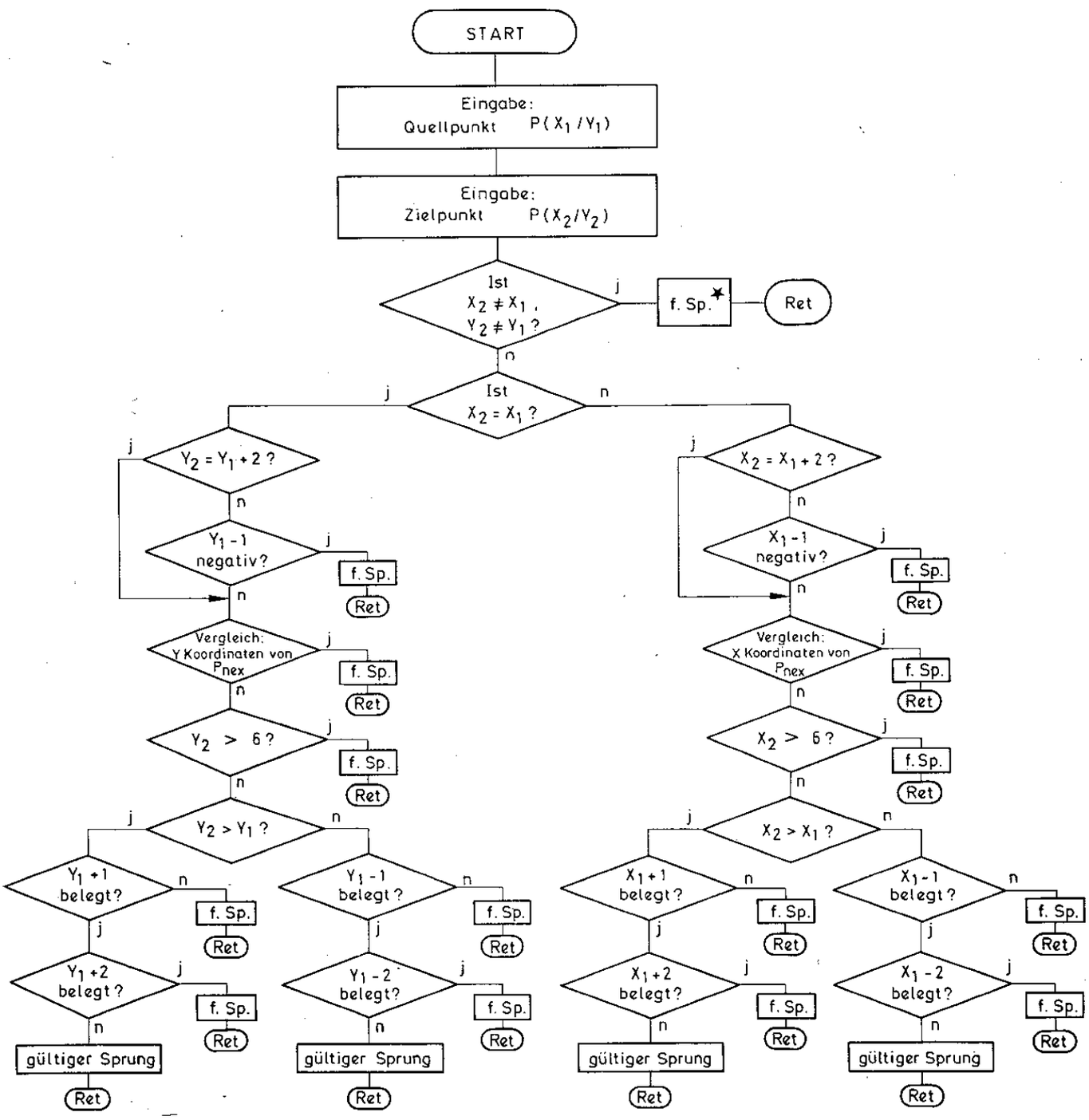
Bild 6.31

Sprung im Spielfeld abgefragt werden müssen. Wie aus der Auflistung der Bedingungen schon zu ersehen ist, erfordert die Prüfung einen vielfachen Zeichenkettenvergleich.

Das Programm beginnt mit der Eingabe des Quell- und Zielpunkts (getrennt in x- und y-Koordinaten). Die Spielfigur, die man zu verlagern wünscht, ist damit im Computer erfaßt. Programmtechnisch ist es von Vorteil, für die Koordinatenwerte Labels als Speicheradressen zu definieren, z.B. ZIEL X, ZIEL Y, QUELLE X, QUELLE Y. Die erste Verzweigung bezieht sich darauf, ob die horizontale oder vertikale Sprungrichtung eingehalten wird, d.h. der Test der x- und y-Koordinaten des Zielpunkts muß ergeben, daß einer von beiden unverändert geblieben ist. Ist dies nicht der Fall, ist bereits hier der Sprung der Spielfigur als ungültig erkannt worden. Programmtechnisch kann der Test sehr einfach mit zwei Subtraktionen erledigt werden.

$x_2 - x_1$ → SUBB RH 2 , QUELLE X → Flags (Z)
$y_2 - y_1$ → SUBB RL 2 , QUELLE Y → Flags (Z)

Durchführung eines Sprungs im Spielfeld



*falscher Sprung

Bild 6.32

Als Programmausschnitt wäre die Verzweigung folgendermaßen lösbar:

Zeile	Mnemonic	Bezeichnung	Bemerkung
1	LDB	RH 2 , ZIEL X	! Ziel-Koordinate x_2 laden !
2	SUBB	RH 2 , QUELLE X	! Quell-Koord. x_1 subtrahieren !
3	JP	EQ , PROGRAMM	! Fortsetzung der Routine !
4	LDB	RL 2 , ZIEL Y	! Ziel-Koordnate y_2 laden !
5	SUBB	RL 2 , QUELLE Y	! Quell-Koord. y_1 subtrahieren !
6	JP	EQ , PROGRAMM	! Fortsetzung der Routine !
7	JP	AUSGABE	! springe in Ausgaberoutine "falscher Sprung" !

Der Programmteil besagt, daß nach dem Test der Koinzidenz eines Koordinatenpunktes die erste Verzweigung bereits erledigt ist, war $x_2 = x_1$. War schließlich $y_2 \neq y_1$, erfolgt ein Sprung in die Ausgaberoutine. Diese Routine zeigt dem Spieler an, daß er einen falschen Sprung ausgeführt hat.

Die zweite Verzweigung ergründet nun genau die horizontale oder vertikale Sprungrichtung. Ist $x_2 = x_1$, muß der Sprung in vertikaler Richtung erfolgt sein. An dieser Stelle erfolgt die Trennung in zwei nicht mehr zusammenführende Wege, die die ausschließliche Behandlung vertikaler oder horizontaler Koordinatenwerte weiter verfolgen.

Die folgenden beiden Verzweigungen legen die Sprungweite fest, die als Ziel innerhalb einer Spielfeldzeile oder -spalte angewählt wurde. Die Verzweigung, z.B. $y_2 = y_1 + 2$?, beinhaltet nicht nur den Test, ob $y_1 + 2$ eingegeben wurde, gleichsam mit einer gültigen Koordinatenänderung zu verstehen; es wird ebenso überprüft, ob nicht mehr als +2 Stellen gesprungen wurde. Wurde dagegen verstoßen, wird letztlich wieder mit

```
JP   AUSGABE
```

gehandelt. Der Test $y_1 - 1$ stellt einfach fest, ob das MSB des neuen Koordinatenwerts gleich 1 ist (negativer Wert). Diese Abfrage erfolgt mit dem Befehl

```
TSET   ZIEL X
JP   MI, AUSGABE
```

oder

```
TSET   ZIEL Y
JP   MI, AUSGABE
```

Der Befehl TSET (Test and Set S-Flag, MSB des Zieloperanden) ermöglicht eine schnelle Prüfung eines neuen Koordinatenwerts. Würde man mit SUBB verfahren, wie vorhergehend bereits gezeigt wurde, setzt gleich anschließend nach der Subtraktion ein bedingter Sprung im Programm ein

```
JP   MI, AUSGABE
```

wenn das Ergebnis negativ war.

Die nächste Verzweigung betrifft einen Vergleich zu den 16 illegalen Koordinatenpunkten, die innerhalb des 6 x 6-Rasters liegen. Diese 16 Randpunkte wurden mit ihren Koordinaten bereits in der Zusammenstellung der Bedingungen genannt. Der Test des Zielpunktes bezüglich seiner Koordinaten zu den illegalen Randpunkten wird mit dem Befehl CPIRB durchgeführt. Dieser Befehl ist praktisch identisch mit dem Z 80-Befehl CPIR, der einen Wert im Speicher (adressiert durch das HL-Register) mit dem Inhalt im Akkumulator vergleicht. Das BC-Register arbeitet als Zählerindex und wird nach jedem Vergleich dekrementiert. Entweder wurde ein gleicher Wert gefunden (A = (@ HL) → Speicherzelle) oder BC wurde auf Null dekrementiert. In beiden Fällen wird die Befehlsausführung beendet, bzw. abgebrochen. Der Z 80 ist einer der wenigen 8-bit-Prozessoren, die diesen Stringvergleich in

einem Befehl wiederholend ausführen. Wenn dies bereits der Z 80 bietet, ist es nur logisch, wenn der Z 8000 hier noch einiges zulegen kann. Grundsätzlich übertrifft CPIRB des Z 8000 den Befehl CPIR des Z 80 durch die individuelle Programmierbarkeit der Bedingung. Für die 16 Randpunkte eines illegalen Ortes im Spielfeld wird man den Z 8000-Befehl folgendermaßen schreiben (Beispiel):

CPIRB RH 6,@R 1, R 12, NE

Im Byte-Register RH 6 steht die Zielkoordinate von X, bzw. Y (ZIEL X, oder ZIEL Y). Dem Vergleich-Befehl muß zuerst ein Lade-Befehl vorausgehen, weil RH 6 bei der Befehlsausführung direkt adressiert wird, d.h. der wirklich zu vergleichende Wert in Register RH 6 stehen muß. Somit wird RH 6 geladen

LDB RH 6 , ZIEL X

Der Programmausschnitt für den Test der Randpunkte (Verzweigung P_{nex}) lautet folgendermaßen:

Zeile	Mnemonic	Bezeichnung	Bemerkung
1	LDB	RH 6 , ZIEL X	! Koordinate X in RH 6 laden !
2	LD	R 1 , RANDPUNKT	! Startadresse für Randpunkt laden !
3	LD	R 12 , #16	! Anzahl der Randpunkte laden !
4	CPIRB	RH 6 , @R 1, R 12, NE	! Vergleich !
5	JP	Z , AUSGABE	! Ausgaberoutine "falscher Sprung", wenn Z = 1 war !

ZIEL X ist eine direkte Speicheradresse für die neue X-Koordinate. Nachdem RH 6 mit dem Inhalt geladen wurde, muß die Startadresse für die Randpunkte in R 1 initialisiert werden. In R 12 wird die Anzahl der zu vergleichenden Punkte (16 Randpunkte) geladen. Sodann kann der Vergleich erfolgen. CPIR ist programmtechnisch eine Subtraktion, wie sie bei vorhergehenden Verzweigungen diskret angewendet wurde. Der Programmcode des CPIRB-Befehls ist in **Bild 6.33** dargestellt. Der Befehl wird aus zwei 16-bit-Wörtern gebildet. Die im W/B-Feld angegebene Null besagt, daß es sich um die Byte-Ausführung des Befehls handelt. Im gezeigten Fall ist der Inhalt von RH 6 das Vergleichsobjekt. Das dst-Feld bezeichnet mit 0110 dieses Byteregister. Im src-Feld wird das als Indexregister benutzte Wortregister R 1 mit 0001 bezeichnet. Der Inhalt von R 1 liefert die Adresse für einen Randpunkt. Er wird nach einem Vergleich um 1 erhöht. Es sei nocheinmal darauf hingewiesen, daß **Register R 0 keinesfalls als Indesregister** verwendet werden darf. Das Zählerregister R 12 wird mit 1100 im r-Feld angewählt. Er wird nach einem Vergleich um 1 erniedrigt. Das cc-Feld schließlich beinhaltet die Bedingung. In einer Tabelle wurde der Bedingungscode in Abschnitt 3.3 erklärt. Für das Spielprogramm wird NE (= not equal) eingesetzt. Dabei ist das Z-Flag betroffen, das nach einem Vergleich immer ungleich Null sein soll. Würde es Eins werden, wäre damit ein illegaler Sprung im Spielfeld erkannt worden. Folglich erklärt sich daraus der Sprungbefehl JP in Zeile 5 mit der Bedingung Z (gleich 1). Der Programmzähler des Z 8000 wird mit der Anfangsadresse der Ausgaberoutine geladen, wenn Z = 1 war. Andernfalls wird im Programm weitergearbeitet.

Die syntaktisch richtige Schreibweise des CPIRB-Befehls ist in Assembler in Zeile 4 des obigen Programmausschnitts angegeben. Der Opcode in Bild 6.33 stimmt hier genau mit dieser Schreibweise überein, wollte man in HEX programmieren. **Bild 6.34** zeigt mit einer Eins im W/B-Feld einen winzigen Unterschied an. Es betrifft die Wortausführung des Befehls. Syntaktisch wird der

Befehl

CPIR R 6 , @ R 1 , R 12, NE

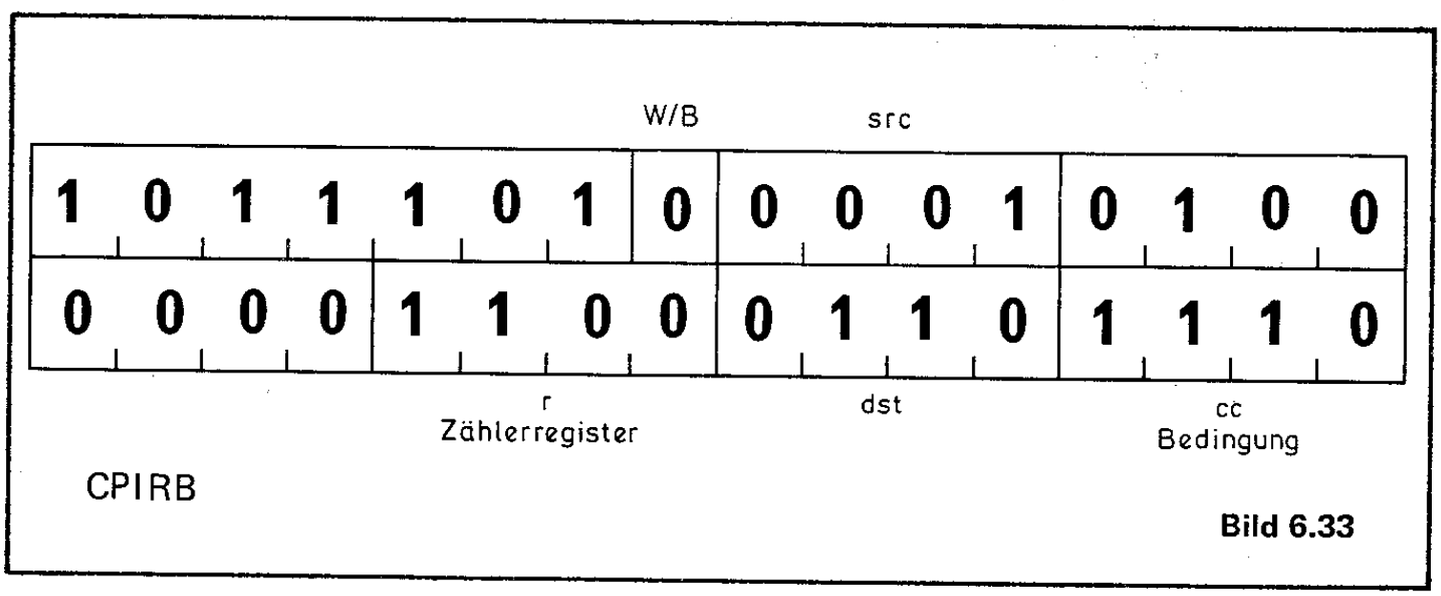
in Assembler geschrieben. Als Maschinencode steht der Befehl beispielsweise mit folgenden Adressen im Arbeitsspeicher:

Adresse (HEX)	Byte (HEX)	
01F0	BB	oberes Byte (gerade Adresse)
01F1	14	unteres Byte (ungerade Adresse)
01F2	0C	oberes Byte (gerade Adresse)
01F3	6E	unteres Byte (ungerade Adresse)

Das obere Byte (D 8 . . . D 15) steht immer an einer geraden Adresse, wie es in Kapitel 2 und 4 mehrfach erwähnt wurde. Für den gegebenen Fall benötigt die CPU eine Ausführungszeit von 155 Zyklen (11 + 9 · 16), gleich 38,75 µs, wenn alle 16 Randpunkte bezüglich X- oder Y-Koordinaten verglichen werden. Maximal wären 65536 Bytes oder 32768 Worte mit einem Byte bzw. Wort vergleichbar.

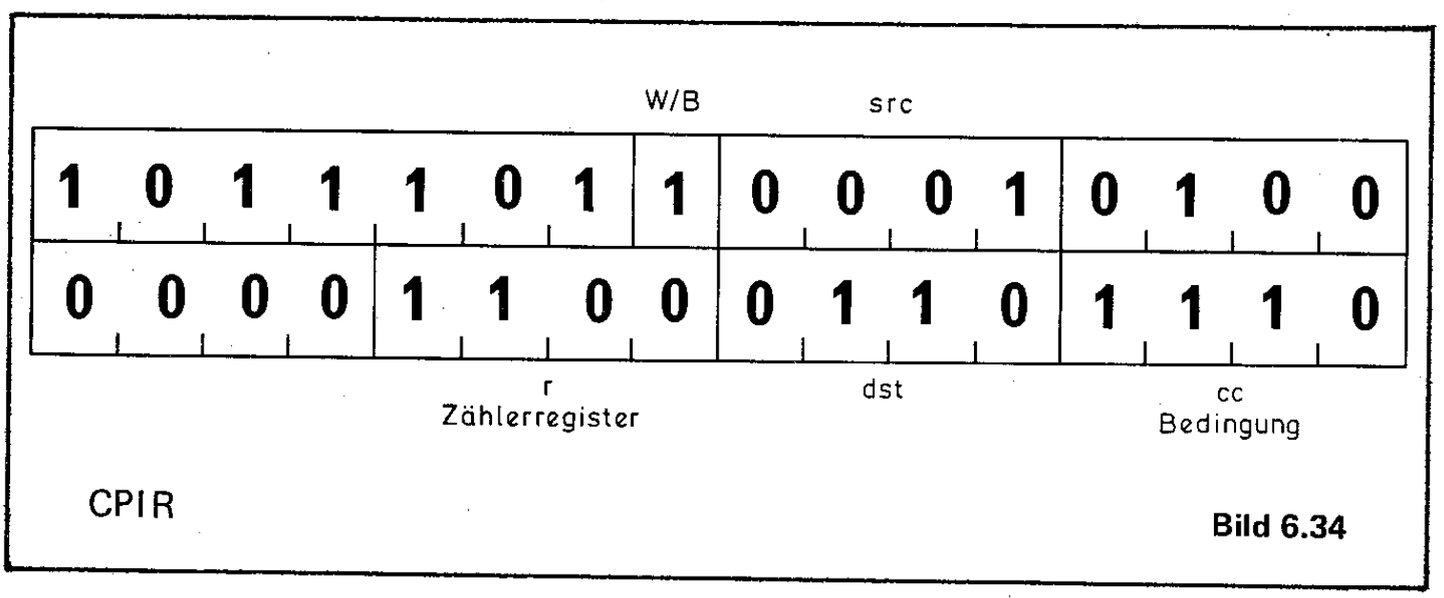
Es gibt noch eine weitere Möglichkeit eines Vergleichs. Der Befehl CPSIR bewirkt den Vergleich **zweier** im Arbeitsspeicher stehender Bytes oder Worte. Beide Operanden werden indirekt adressiert. In **Bild 6.35** ist zum Vergleich des vorhergehenden Befehls der Opcode von CPSIR (Wortausführung) dargestellt. CPSIR hat nur an der Stelle 2_1 des ersten Wortes eine Eins statt einer Null stehen, wenn man die Verwendung gleicher Register wie der Bedingung voraussetzt. Die syntaktische Schreibweise lautet:

CPSIR @ R 6 , @ R 1, R 12, NE



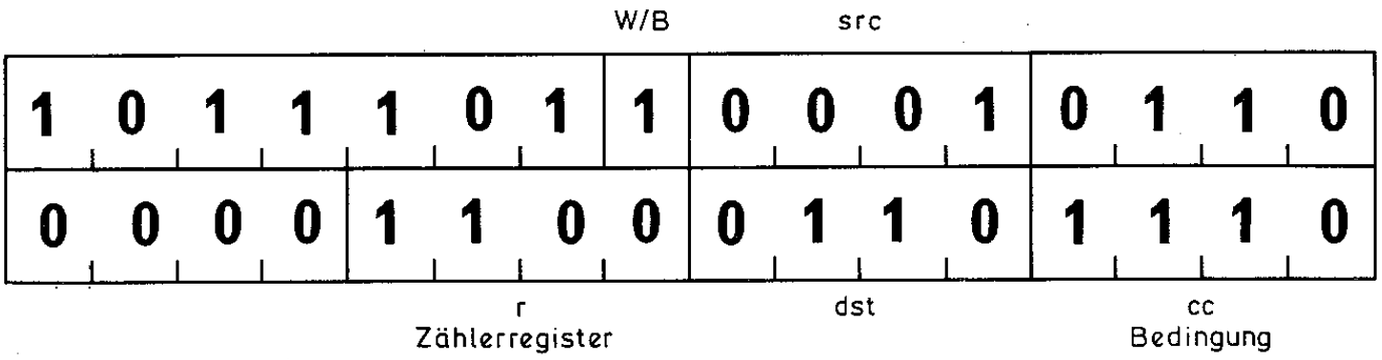
CPIRB

Bild 6.33



CPIR

Bild 6.34



CPSIR

Bild 6.35

Nach der Ausführung eines Vergleichs werden R 6 und R 1 um je 2 Stellen erhöht und R 12 um eine Stelle erniedrift. Die Anwendung dieses Befehls ist jedoch zeitintensiver als CPIR und benötigt 32 zusätzliche Bytes mehr Arbeitsspeicher. Für das Spiel ist dieser Stringbefehl zwar anwendbar, jedoch nicht sinnvoll.

Die folgenden Verzweigungen im Flußdiagramm nach Bild 6.32 spezifizieren weitere Forderungen für einen gültigen Sprung. $x_2 > 6$ bzw. $y_2 > 6$ bezeichnen bei einer Erfüllung einen Sprung über den oberen bzw. rechten Spielfeldrand und damit einen illegalen Sprung. Die programmtechnische Lösung ist wiederum sehr einfach zu bilden.

Zeile	Mnemonic	Bezeichnung	Bemerkung
1	LDB	RH 0 , ZIEL X	! laden Zielkoord. x in Byteregister !
2	SUBB	RH 0 , #7	! subtrahiere 7 von Byteregister !
3	JP	PL , AUSGABE	! falscher Sprung, wenn Ergebnis pos. !
4	JP	Z , AUSGABE	! falscher Sprung, wenn Ergebnis null !

Die Routine ist für den Test der y-Koordinate gleichartig aufgebaut. Die Verzweigung $x_2 > x_1$ bzw. $y_2 > y_1$ legt nun eindeutig fest, ob vertikal nach oben oder unten bzw. horizontal nach rechts oder links gesprungen wurde. In den letzten beiden Verzweigungen, getrennt in die Sprungrichtung, wird nur noch festgestellt, ob die Nachbarplätze mit einer Spielfigur belegt sind. Für diese Operation ist ebenfalls der CPIR-Befehl anwendbar. Es wird in einer Tabelle gesucht, ob die Koordinatenstelle (Spielfeld) belegt ist oder nicht. Dies ist ein Vergleich von Werten, wie er bereits beschrieben wurde. Ist zuletzt festgestellt worden, daß z.B. $x_1 + 2$ nicht belegt ist, war der Sprung im Spielfeld gültig. Als Beispiel wird ein Zweig des Flußdiagramms programmtechnisch erläutert.

Die Adresse % SSXX ist die Anfangsadresse für die 33 Werte mit den Koordinaten für einen belegten Spielfeldpunkt. Die Adresse % SSYY ist die Anfangsadresse für die 33 Werte mit den Koordinaten für einen freien Spielfeldpunkt.

Wird ein CPIRB-Befehl abgebrochen, weil das betreffende Flag gesetzt wurde, wird anschließend durch einen bedingten JP in die AUSGABE für einen "falschen Sprung" gesprungen. Wurde CPIRB regulär beendet (R 2 gleich Null), wird normal weitergearbeitet.

Zeile	Mnemonic	Bezeichnung	Bemerkung
1	LDB	RH 0 , QUELLE X	! laden Quellkoord. in Byteregister !
2	ADDB	RH 0 , #1	! Quellkoord. um 1 erhöhen !
3	LD	R 1 , #% SSXX	! Startadresse für Tabelle "belegt" !
4	LD	R 2 , #33	! Anzahl der Tabellenwerte !
5	CPIRB	RH 0 , @R 1, R 2, NE	! Vergleich !
6	JP	Z , AUSGABE	! falscher Sprung, wenn Z = 1 !
7	ADDB	RH 0 , #1	! Quellkoord. um 1 erhöhen !
8	LD	R 1 , #% SSYY	! Startadresse für Tabelle "frei" !
9	LD	R 2 , #33	! Anzahl der Tabellenwerte !
10	CPIRB	RH 0 , @R 1, R 2, NE	! Vergleich !
11	JP	NZ , AUSGABE	! falscher Sprung, wenn Z = 0 !
12	JP	SPRUNG	! Ausgabe: gültiger Sprung !
13		END Procedure	

Der Vorteil der CPIR(B)-Operation ist, daß das Zielregister (in diesem Fall RH 0) unverändert bleibt. Das vereinfacht die Programmierung wesentlich. Diese Eigenschaft gilt auch für den nur Bytebefehl CPIR der Z 80-CPU.

Nachdem die letzte Verzweigung behandelt und positiv abgeschlossen wurde, beginnt mit JP SPRUNG die Ausgabe des anerkannten Zielpunktes P mit dem Hinweis: "gültiger Sprung übernommen". Als Folgeroutine werden die Veränderungen von Spielfiguren in die beiden Tabellen (je 33 Werte) für freie und belegte Spielfeldpunkte übernommen. Eine neue Eingabe kann darauf wieder erfolgen.

Als weitere Stringroutine kann man beispielsweise den richtigen Weg zur Lösung eingeben. Ein Lösungsweg mit dem angegebenen Zwischenstand nach dem 17. Sprung in Bild 6.30 benötigt 31 Sprünge. Die Reihenfolge der Sprünge wird in Form einer Tabelle festgelegt und in den Speicher eingegeben. Der Test des neu eingegebenen Zielpunktes erfolgt wieder mit dem Befehl CPIR(B) mit der Bedingung EQ (equal). Somit kann der neue Zielpunkt nicht nur auf Gültigkeit nach Bild 6.32, sondern auf die erfolgreiche Reihenfolge hin getestet werden.

Die Programmierung des Solitaire-Spiels ist mit dem Z 8000-Befehlssatz so einfach wie noch nie auf der Assemblerebene. Da die beschriebene Programmierung exemplarischen Wert besitzt, wird an dieser Stelle kein komplettes Listing angegeben. Ein funktionierendes Programm liegt beim Autor in der Schublade. Es ist auf besonderen Wunsch zugänglich. Es sei jedoch bemerkt, daß ein funktionierendes Programm an Hand der gegebenen Beispiele sehr leicht selbst zu entwerfen ist. Mit dem Z 8000 ist das auch keine Kunst mehr.

6.4 EIN MATHEMATISCHES BEISPIEL: SCHNELLE FOURIER-TRANSFORMATION

Ein Computerbaustein der Klasse eines Z 8000 wird künftig für Aufgaben eingesetzt werden, die mit einem 8-bit-Prozessor softwaremäßig nur mit einem unverhältnismäßig hohen Aufwand lösbar sind oder wegen entsprechenden zeitli-

chen Anforderungen überhaupt nicht erfüllt werden können. Viele Anwender kennen das Problem. Die Hersteller haben sich schon vor der Entstehung der "16 bitter" dem Problem angenommen. Man versucht Abhilfe zu schaffen, indem spezifische "Mathematiker" (sprich: Hochleistungs-Mathematik-Prozessoren) wie beispielsweise der Am 9511 und der Am 9512 zusammen mit 8-bit-CPUs eingesetzt werden. Da diese Prozessoren durchweg sehr teuer sind und überdies einige zusätzliche Hardware notwendig ist, erweist sich eine Erweiterung eines 8-bit-Systems auf diese Art nicht unbedingt als Vorteil. Grundsätzlich wird in all den Aufgaben, die umfangreiche Berechnungen erfordern, entweder die genannte Methode notwendig oder von vorneherein ein Z 8000-System zum Einsatz kommen. Noch vor kurzer Zeit gab es hierfür keine Alternative. Mit dem Z 8000 gibt es eine ernsthafte, sehr wirtschaftliche Alternative. Gemeint sind die Anwendungen im Bereich der schnellen Signalanalyse.

Wie man weiß, bedingt eine Signalanalyse die Anwendung der Fourier-Transformation. Wer sich in diesem Bereich etwas auskennt, der weiß um die Quantität an Berechnungen (Reihenentwicklungen numerisch). Um hier in den jeweiligen Fällen eine softwaremäßig vertretbare Lösung ausarbeiten zu können, muß der Prozessor über die vier Grundrechnungsarten verfügen. Der Z 8002 ist alles in einem; Mathematik- und Standard-Prozessor. Als konkretes Beispiel sei noch einmal das kleine System in Bild 4.72 genannt. Dieser auf einer Doppel-Europaplatine aufzubauender Rechner ist bereits für eine Online-Signalanalyse vollkommen ausreichend. Die CPU berechnet die einzelnen Fourier-Koeffizienten schnell genug und bedient mit ihren Ergebnissen die peripheren Systeme über die Schnittstellen. Ein A/D- und D/A-Umsetzer bzw. die nötigen Skalierungs- und Halteverstärker hätten auf der Platine auch noch Platz. Bisher war dies mit einer so minimalen Bausteinzahl nicht möglich.

Der Einsatz der Signalanalyse reicht von speziellen regelungstechnischen Anwendungen bis zur Spracherkennung. Ein Sprachsignal f(t) wird über eine Kurzzeit-Spektralanalyse nach den einzelnen Frequenzanteilen erfaßt bzw. in ein Amplituden- und Phasenspektrum über der Frequenz zerlegt. Der Computer wird als Klassifikator eingesetzt, bestimmt aus dem Sprachsignal f(t) innerhalb eines Zeitfensters die Frequenzanteile einer Gewichtsfunktion h(t). Die Funktion h(t) wird aus einer Übertragungsfunktion eines Filters abgeleitet (Sprungantwort eines Filters). Die Filterausführung ist überlicherweise analog, wird jedoch schon von digitalen Filtern ersetzt.

Die physikalische und mathematische Grundlage bildet die diskrete Fourier-Transformation DFT, die man als die klassische, nicht computergerechte Art bezeichnet. Der grundlegende Zusammenhang jeder Signalanalyse mit dem Computer wird aus dem Fourier-Integral der komplexen Form abgeleitet. Man unterscheidet zwischen der Spektralfunktion $F(\omega)$ und der nicht periodischen Zeitfunktion f(t), die sich in der allgemeinen Form

Spektralfunktion (1)

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t} dt$$

und **Zeitfunktion** (2)

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega$$

darstellen. Das komplexe Spektrum der Zeitfunktion f(t) wird durch das komplexe Spektrum der Amplitudendichte $F(\omega)$ abgebildet. Die grafische Darstellung der komplexen Amplitudendichte wird als Betrag und Phase über ω angegeben. Die Berechnung der Fourier-Koeffizienten erfolgt nach einer Euler-

Fourierschen Formel. Die drei Fourier-Koeffizienten unterscheiden sich in

a_o = Gleichspannungsanteil
a_n = Cosinusglieder
b_n = Sinusglieder

Man muß sich einen beliebigen elektrischen Schwingungsvorgang aus dem Gleichstromanteil mit einer Anzahl einzelner Sinusschwingungen unterschiedlicher Frequenz und Amplitude zusammengesetzt denken, da Cosinusglieder phasenverschobene Sinusglieder sind. Der Amplitudenspitzenwert ergibt sich aus

$$A_n = \frac{\sin \varphi_n}{a_n} = \frac{\cos \varphi_n}{b_n};$$

$$A_n = \sqrt{a_n^2 + b_n^2}$$

Die Division von a_n durch b_n ergibt den Tangens des Phasenwinkels

$$\tan \varphi_n = a_n/b_n$$

Die beiden Grundformen der Fourier-Reihe ergeben sich aus

$$f(t) = \sum_{n=o}^{\infty} (a_n \cos n\omega t + b_n \sin n\omega t) = \sum_{n=o}^{\infty} A_n \sin (n\omega t + \varphi_n) \qquad (3)$$

Die Grundschwingung wird mit $A_1 \sin(\omega t + \varphi_1)$ angegeben. Üblicherweise wird die Zeitfunktion f(t) in den Grenzen 0 . . . 2π betrachtet. Dies ist insbesondere dann der Fall, wenn die Funktion periodisch ist, weil der Bereich 2π die Funktion charakterisiert. Sie ist damit stellvertretend erfaßt. Liegt keine Periodizität der Zeitfunktion vor, wird die Reihenentwicklung grundsätzlich nur im Bereich 2π erfaßt. Das konstante Glied a_o beeinflußt die Periodizität nicht. Die Kurve wird nur in Richtung der Ordinate verschoben. Um also eine Zeitfunktion in eine Fourier-Reihe zu entwickeln, müssen die Koeffizienten a_o, a_n, b_n und ferner A_n und φ_n in der Betrachtungsgrenze 2π berechnet werden. Für die drei Hauptkoeffizienten gelten die Formeln

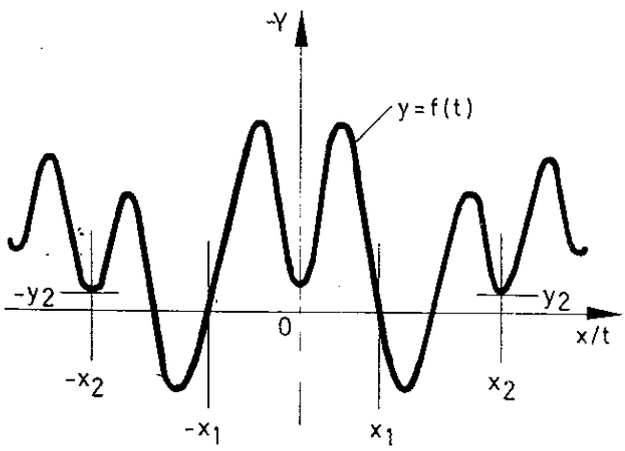
$$a_o = \frac{1}{2\pi} \int_o^T f(t)\omega dt = \frac{\omega}{2\pi} \int_o^T f(t)\, dt; \qquad \left[\frac{2\pi}{T} t = \omega t\right]$$

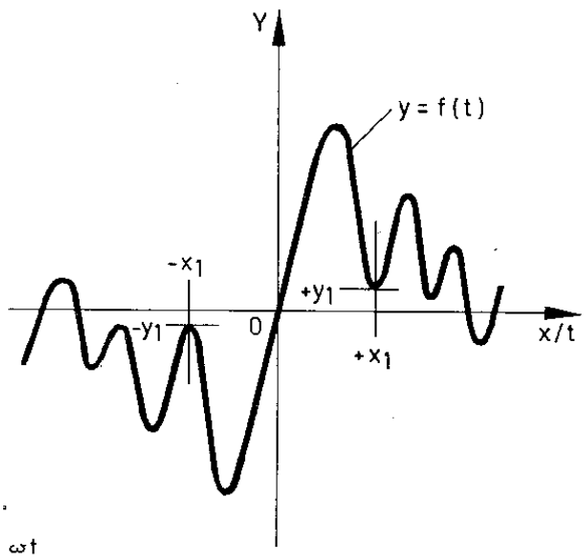
$$a_n = \frac{1}{\pi} \int_o^T f(t)\omega \cos n\omega dt = \frac{2}{T} \int_o^T f(t) \cos n\omega t\, dt$$

$$b_n = \frac{1}{\pi} \int_o^T f(t)\omega \sin n\omega dt = \frac{2}{T} \int_o^T f(t) \sin n\omega t\, dt$$

Man unterscheidet bei einer Zeitfunktion allgemeiner periodischer Art zwischen geraden und ungeraden Funktionen. **Bild 6.36** zeigt den prinzipiellen Unterschied. Der Gleichanteil a_o ist bei ungeraden Funktionen generell gleich Null. Das Integral zur Bestimmung von a_o bezieht sich auf den Flächeninhalt

Periodische Funktionen





$$x = \frac{2\pi}{T} = \omega t$$

$$dx = \omega\, dt$$

achsensymetrische gerade Funktion

$$y = f(t) = f(-t)$$

$$a_0 = \frac{2}{T}\int_0^{T/2} f(t)\,dt$$

$$a_n = \frac{4}{T}\int_0^{T/2} f(t)\cos n\omega t\,dt$$

$$b_n = 0$$

zentralsymetrische ungerade Funktion

$$y = f(t) = -f(-t)$$

$$a_0 = 0$$

$$a_n = 0$$

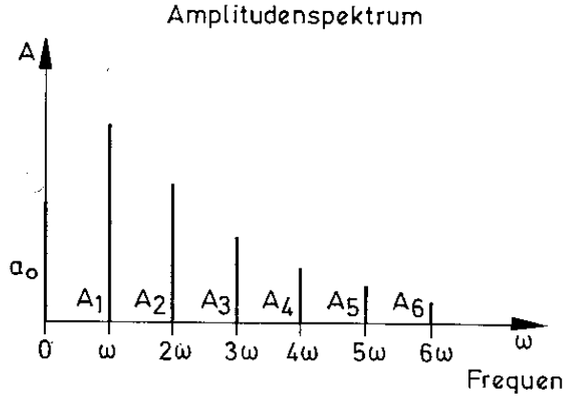
$$b_n = \frac{4}{T}\int_0^{T/2} f(t)\sin n\omega t\,dt$$

Darstellung
(Sinusreihenentwicklung)

$$f(t) = \sum_{n=0}^{\infty} A_n \cdot \sin(n\omega t + \varphi_n) =$$

$$= A_0 \cdot \sin\varphi_0 + A_1 \cdot \sin(\omega t + \varphi_1) + A_2 \cdot \sin(2\omega t + \varphi_2) + \ldots$$

Amplitudenspektrum



Phasenspektrum

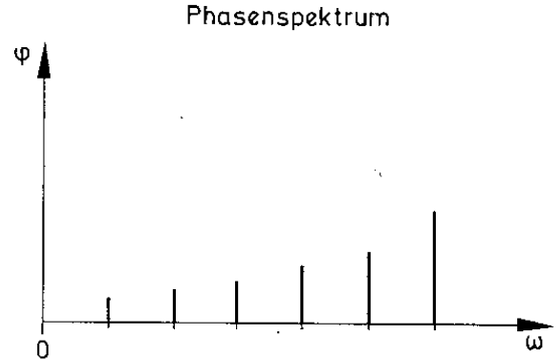


Bild 6.36

der ungeraden Funktion, deren zentralsymmetrische Anordnung eine Aufhebung der Flächen oberhalb und unterhalb der Abszisse bewirkt. Der Gleichanteil liegt somit direkt auf der Abszisse.

Ein sehr wichtiger Punkt bei der Anwendung der Fourier-Transformation ist, daß der nicht-sinusförmige Schwingungsvorgang aus einem linearen System entstanden sein muß. Stammt die Zeitfunktion aus einem nichtlinearen Übertragungsglied, treten Beeinflussungen

der einzelnen Oberwellen untereinander auf. Nichtlinear bedeutet, daß der mathematische Zusammenhang zwischen Eingangs- und Ausgangsgröße nicht mehr linear proportional ist. Als Beispiel ist ein übersteuerter Verstärker oder ein nichtlinearer Meßfühler zu nennen.

Die bisherige Betrachtung der Fourier-Transformation waren Grundlagen. Wie schon gesagt, ist die diskrete FT (DFT) nicht besonders gut für den Computer geeignet, weil die Lösung von Integralen nach numerischen Verfahren (Simpson' sche Regel, Gregorysche Integrationsformel. Koeffizientenvergleich) für den Computer nicht gerade einfach aufzubereiten sind. Der Zuschnitt einer Fourier-Transformation für den Computer, insbesondere beim Einsatz in schnellen Regelprozessoren, muß an der DFT-Formel

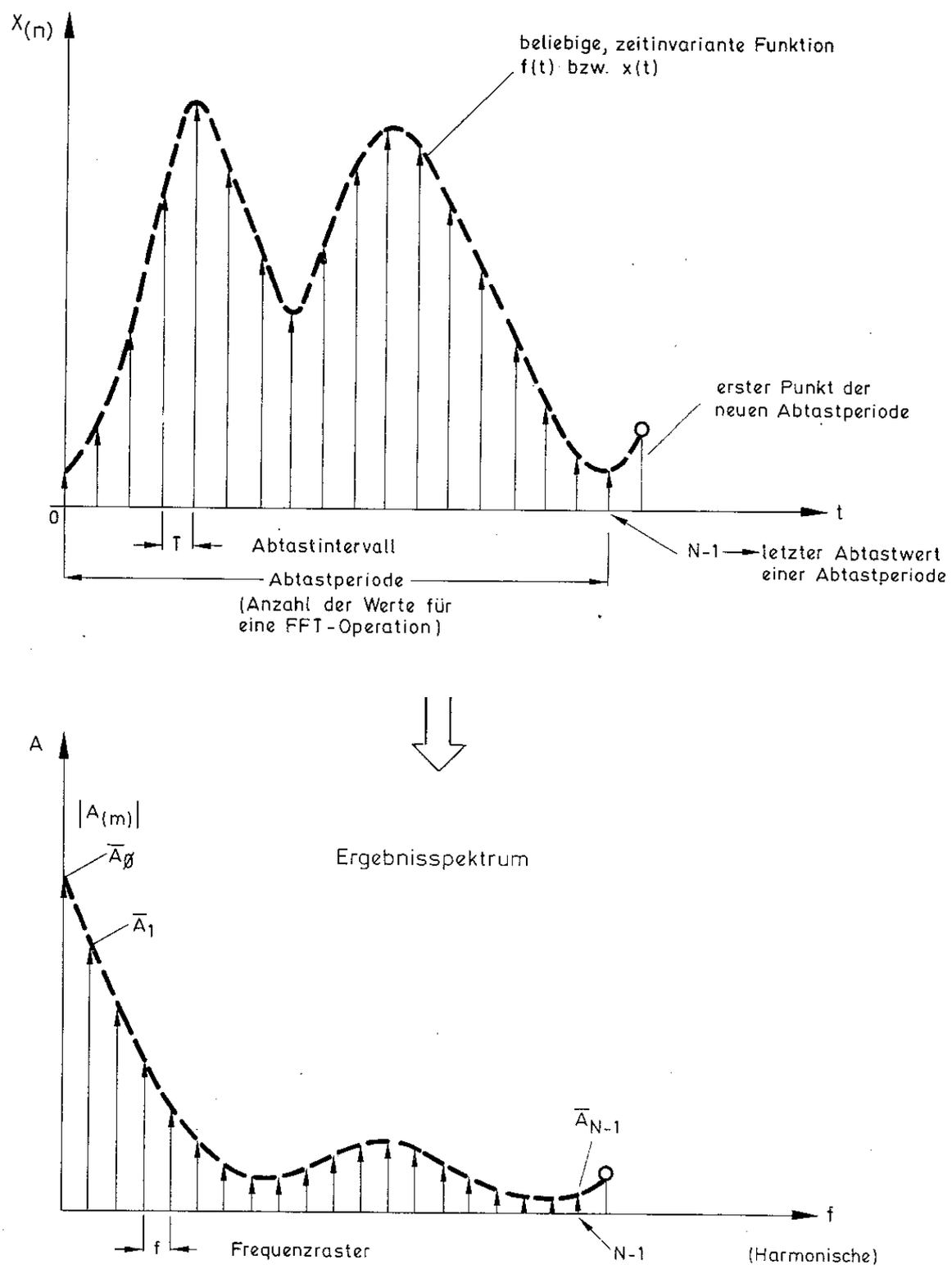


Bild 6.37

selbst ansetzen. Es entstand die schnelle Fourier-Transformation (FFT = Fast Fourier Transform). Die FFT ist ein schneller Algorithmus, der aus der DFT abgeleitet wurde und deshalb so schnell ist, weil er die vielfachen Redundanzen der DFT-Formel nicht mehr enthält. Das ist grundsätzlich und nicht CPU-abhängig. Folgender Zusammenhang soll dies erläutern. Das erwünschte Ziel, aus einem gegebenen mathematischen Verhalten eines Vierpols ein Übertragungsspektrum zu ermitteln, ist in **Bild 6.37** dargestellt. Aus einer beliebigen, zeitinvarianten Funktion (im Bild f(t) bzw. x(t)) bezeichnet, soll eine komplexe Amplitudendichte (Ergebnisspektrum) ermittelt werden. Ein Wertesatz von N − 1 Abtastwerte beschreiben die Funktion f(t). Das Abtastintervall ist mit T gegeben. Das Ergebnisspektrum soll nun die Zahl N − 1 Amplituden enthalten, angefangen vom Gleichanteil A_0, der ersten Harmonischen A_1 (= Grundschwingung mit der Frequenz f), der zweiten Harmonischen A_2 (= Oberschwingung mit 2f) usw. bis zur N−1-ten Harmonischen, die f(t) bilden, wenn sie alle gemeinsam überlagert werden.

Der Wertesatz, übernommen durch einen Analog-Digital-Umsetzer, wird aus der Fourier-Reihe (3) durch Umwandlung der Formel in die komplexe Form beschrieben

$$\overline{F}(j\omega) = \sum_{n=o}^{N-1} c_n e^{-j2\pi fn/N}; \quad f\ (\text{Frequenz}) = 0, 1, 2 \ldots, N-1$$

$\overline{F}(j\omega)$: konjugiert komplexe Form der Spektralfunktion

c_n : n-ter Abtastwert der Zeitfunktion

$$c_n(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\omega t)\omega e^{-jn\omega T}\, dt; \quad n = 0, \pm 1, \pm 2 \ldots$$

Die Berechnung der komplexen Produkte ist ein Maß des Rechenaufwandes bei der Bestimmung des Wertesatzes $\overline{F_o}, \overline{F_1}, \overline{F_2} \ldots \overline{F_{N-1}}$. Die gesamte Berechnung des Wertesatzes erfordert N Summen bezüglich der Frequenz f. Jede Summe hat N Produkte aus c_n und der Exponentialfunktion. Die Monotonie der Exponentialfunktion $c_n(t) \cdot e^{-jn\omega t}$ (= $c_n(t) \cdot e^{-j2\pi fn/N}$) beinhaltet eine Reihe redundanter Produkte, die sich im Verlauf der Berechnung des Wertesatzes teilweise mehrfach wiederholen. Die FFT ist keineswegs ein neuer Transformationsmechanismus. Sie ist nur die abgemagerte bzw. optimierte Form der DFT. Ist eine große Zahl von c_n irgendeines dynamischen Vorgangs dem Computer vorgegeben (über den A/D-Umsetzer digitalisiert), ist die FFT die einzige, zeitlich ökonomische Möglichkeit einer Berechnung des Spektrums zu dem Vorgang. Die DFT-Formel (4) wird nur etwas vereinfacht angeschrieben, so daß für jeden $\overline{F}(j\omega)$-Term eine Linearkombination auftritt. Es wird

$$e^{-j2\pi/N} = W_N$$

gesetzt, wodurch die DFT-Formel (4) folgendermaßen aussieht:

$$(5)\quad F(j\omega) = A_m = \sum_{n=0}^{N-1} W_N^{mn} \cdot c_n;$$

$$m = 0, 1, 2, \ldots, N-1$$

Wie aus Bild 6.37 ersichtlich ist, muß man für N einen Wert festlegen, der die Anzahl der Abtastungen von f(t) festlegt. Anzahl N und Abtastintervall T (Sekunden . . . Mikrosek.) werden nach

dem Abtasttheorem ermittelt. Die Abtastwerte c_n sind Nadelimpulsabtastungen von f(t). Für eine sehr gute Reproduktion von f(t) wird mit einer Abtastrate von etwa dem Zehnfachen der höchsten Frequenzkomponente in f(t) gearbeitet. Als vorteilhaft hat sich für den Algorithmus eine Anzahl N herausgestellt, die einem Zweierpotenzwert entspricht. Für die folgend gewählte Anzahl N = 8 ergibt sich für W_8^{mn} eine vielfache Anzahl äquivalenter Potenzen bei der Berechnung der DFT nach (5).

$$W_8^0 = W_8^8 = W_8^{16} = W_8^{24} \ldots$$
$$W_8^1 = W_8^9 = W_8^{17} = W_8^{25} \ldots$$
$$W_8^2 = W_8^{10} = W_8^{18} = W_8^{26} \ldots$$
$$W_8^3 = W_8^{11} = W_8^{19} = W_8^{27} \ldots$$
$$\vdots$$
$$W_8^7 = W_8^{15} = W_8^{23} = W_8^{31} \ldots$$
$$(0 \leqslant m \leqslant N-1)$$

Die FFT beruht auf einer wiederholten, vollständigen Zerlegung des Abtastwertesatzes. Die Zerlegung ist am einfachsten, wenn N als Potenz von 2 angenommen wird. **Bild 6.38** zeigt eine vollständige Zerlegung der Abtastwerte. Die unterste Reihe 0426 . . . 37 zeigt die Reihenfolge der multiplikativen Verknüpfung der acht Werte $c_0 \ldots c_n$. Dies muß nicht so sein; es ist jedoch die einfachste Form einer Verknüpfung. **Bild 6.39** zeigt den prinzipiellen Fall der Verknüpfung aus zwei Abtastwerten. Das Ergebnis (Amplitudendichte) wird in Real- und Imaginärteil getrennt berechnet. Für 8 Abtastwerte kann die Verknüpfung auf zwei Arten geschehen. **Bild 6.40** und **6.41** zeigen die Butterfly-Verknüpfung der Abtastwerte mit W_N zu den konjugiert komplexen Spektralwerten. In Bild 6.40 ist eine Bitumkehrordnung am Eingang und in Bild 6.41 am Ausgang vorgegeben. Die beiden Bilder sind als Flußdiagramm eines Lösungsweges aufzufassen. Das Netz aus Knoten und Pfeilen zeigt die Verknüpfung an, wie die Abtast-

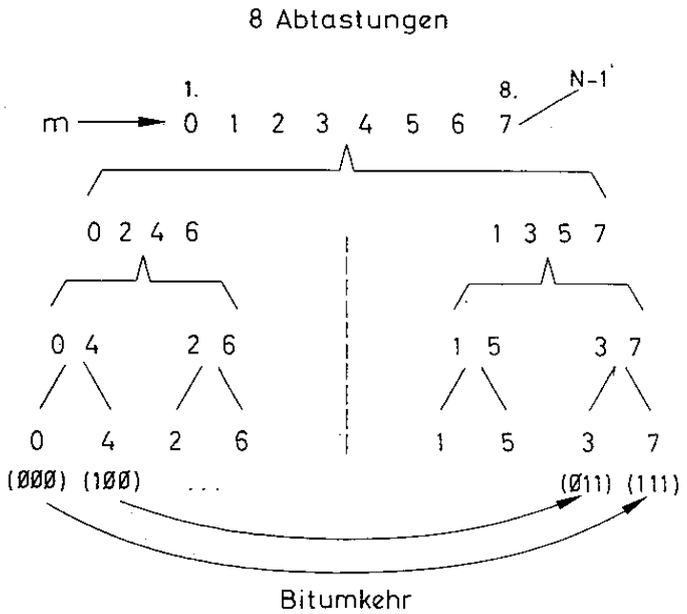


Bild 6.38

werte (c_n bzw. x_n) einen Spektralwert $\overline{A}_m$ ergeben. Das FFT-Butterfly nach Bild 6.40 hat seinen Ursprung nach dem Cooley-Tukey-Algorithmus. Es ist auf niedrige Potenzen optimiert. Jede 1 an einem Pfeil bedeutet $W_8^0 = 1$. In Bild 6.39 zeigen die Einsen an den Pfeilen zu X_1 an, daß dieses Ergebnis nur aus dem Real- bzw. Imaginärteil von x_0 und y_0 besteht. In Y_1 ist W_2^1 und $-W_2^1$ im Real- und Imaginärteil enthalten. Für N = 2

gibt es keine Vereinfachung der DFT durch die FFT. Für N = 8 ist die Ausnutzung von Symmetrien bei den Harmonischen bereits nachhaltig spürbar. Für die Ausführung der FFT nach Cooley-Tukey ergibt sich die Entwicklung einer allgemeingültigen Matrix, die für einen Abtastwertesatz **der Anzahl N in Höhe einer Zweierpotenz** folgendes Schema besitzt:

$$\begin{bmatrix} \bar{A}_0 \\ \bar{A}_1 \\ \bar{A}_2 \\ \bar{A}_3 \\ \bar{A}_4 \\ \bar{A}_5 \\ \vdots \\ \bar{A}_{N-1} \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \dots & W^0 \\ W^0 & W^n & W^{2n} & \dots & W^{(N-1)n} \\ W^0 & W^{2n} & W^{4n} & \dots & W^{2(N-1)n} \\ W^0 & W^{3n} & W^{6n} & \dots & W^{3(N-1)n} \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ W^0 & W^{(N-1)n} & W^{(N-2)n} & \dots & W^{(N-1)^2 n} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \cdot \\ \cdot \\ \cdot \\ c_{N-1} \end{bmatrix}$$

Die Koeffizienten-Matrix W entspricht der DFT für m Elemente. Die Faktorisierung ist mit allen Untermatrizen W_m (für $0 \leqslant m \leqslant N-1$) ausgeführt. Aus der schon gezeigten Äquivalenz der Werte $W_N^m = W_N^{m+N}$ wird die Matrix im praktischen Fall um alle höheren Potenzwerte als N−1 entledigt. Im obigen Fall tritt nur N−1 als größter Exponent auf; es wird nur der Wert n=1 bei der Faktorisierung der Matrix betrachtet. Folglich hat für N=8 und n=1 die Matrix die Form:

$$\begin{bmatrix} \bar{A}_0 \\ \bar{A}_1 \\ \bar{A}_2 \\ \bar{A}_3 \\ \cdot \\ \cdot \\ \cdot \\ \bar{A}_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & \dots & & & & & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ \cdot & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ \cdot & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \cdot \\ \cdot \\ \cdot \\ c_7 \end{bmatrix}$$

Die Faktorisierung gilt für das FFT-Butterfly nach Bild 6.40 (Bitumkehr am Eingang). Nimmt man eine beliebige Zeile heraus und stellt z.B. $\bar{A}_3$ als Summe dar (m = 3, n = 1)

$$A_3 = c_0W^0 + c_1W^3 + c_2W^6 + c_3W^1 + c_4W^4 + c_5W^7 + c_6W^2 + c_7W^5$$

ist die Richtigkeit der DFT nach Formel (4) gegeben.

Praktisch läßt sich die Summe von Koeffizienten aus den einzelnen Signalpfaden in Bild 6.40 ermitteln. Bei genauerer Betrachtung der Koeffizienten des FFT-Butterfly nach Bild 6.40 fallen als größte Potenz von W der Wert $-W^3$ auf. Die

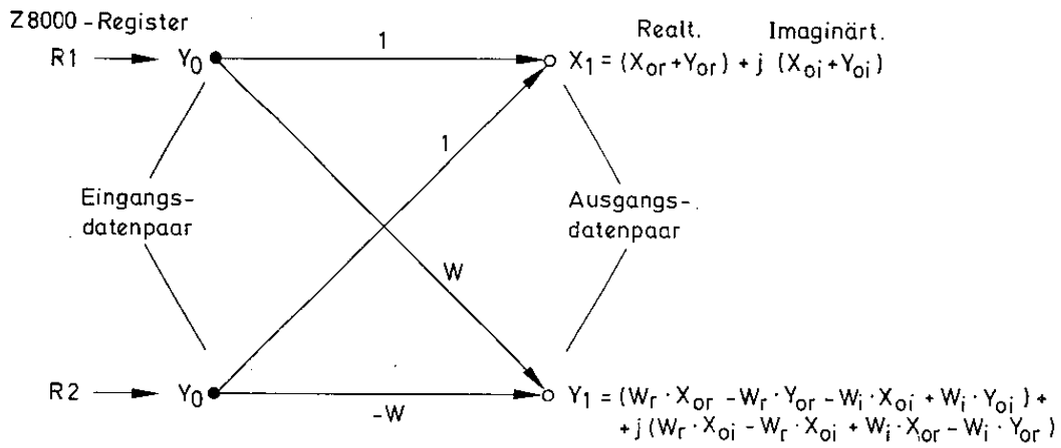
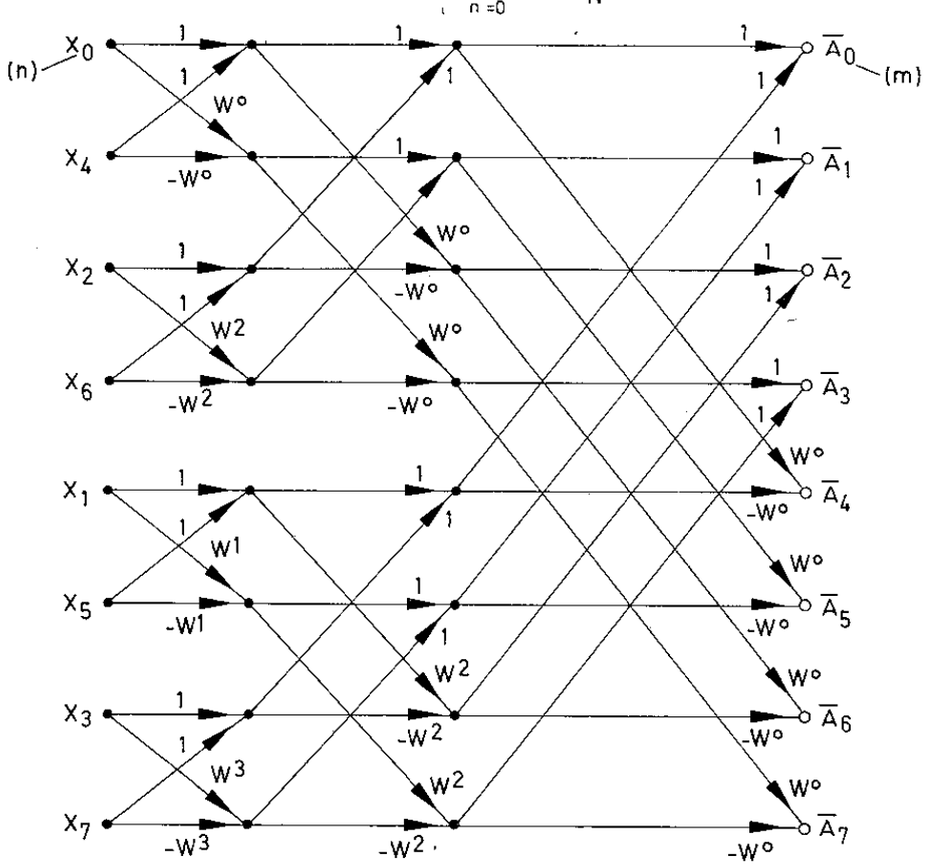


Bild 6.39

FFT - Algorithmus grundsätzlicher Art

$$\overline{A}_m = \sum_{n=0}^{N-1} x(n)\, W_N^{mn}$$



Zeitzerlegung für 8 Abtastwerte (N)
mit Bitumkehrordnung am Eingang

$W_8^o = 1$

$$\left[\overline{A}_{(0\ldots7)} = \sum_{0}^{7} X_{(0\ldots7)}\, W_8^{mn}\right]$$

Bild 6.40

Faktorisierung der Koeffizienten-Matrix W wurde dahingehend ausgerichtet, für den Linearfaktor W die Potenzen zu reduzieren. Es handelt sich um eine korrigierte Form des FFT-Algorithmus, ausgehend von der Faktorisierung, wie sie von H.C. Andrews und K.L. Caspari (IEEE Trans., Januar 1970) beschrieben wurde. Ein Element des Vektors $\bar{A}_m$ wird generell durch die Anwendung des Überlagerungssatzes gebildet. Das Vektorelement $\bar{A}_3$ (vorheriges Beispiel) entsteht aus zwei Pfeilen, die von zwei Zwischenknoten herrühren (Z_3' und Z_7'). Diese beiden Knoten werden wieder von jeweils zwei Zwischenknoten durch Pfeile verbunden. **Bild 6.42** zeigt für $\bar{A}_3$ die Entstehung. Der einfache Zusammenhang lautet:

$$\bar{A}_3 = Z_3' + Z_7' ;$$

$$Z_3' = Z_3 \cdot W^0 + Z_1 \cdot W^0;$$

$$Z_7' = -Z_7 \cdot W^2 + Z_5 \cdot W^2; \text{ usw.}$$

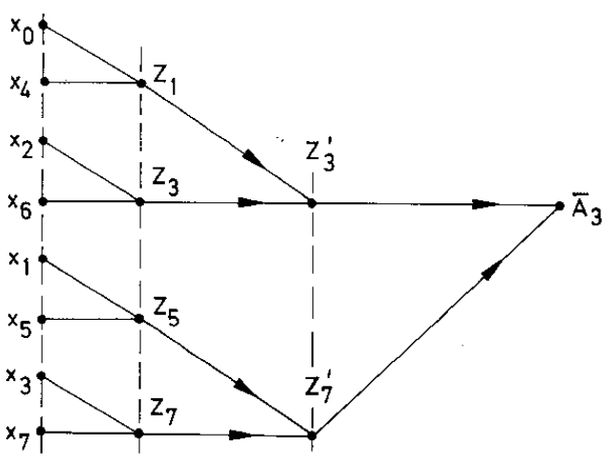
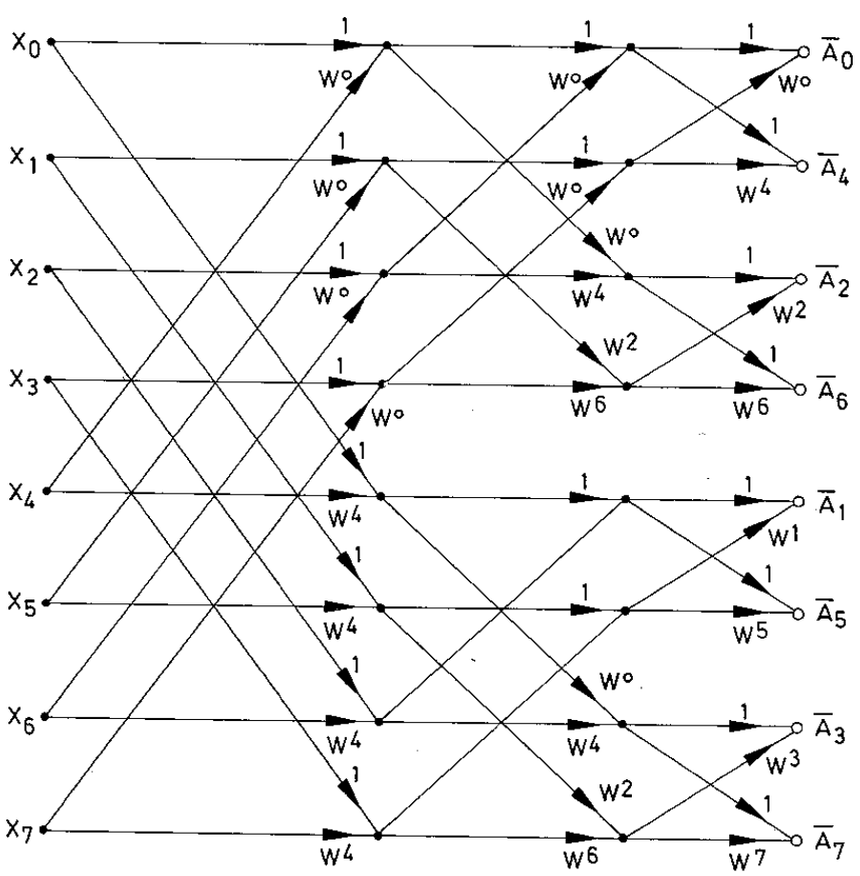


Bild 6.42

FFT - Algorithmus grundsätzlicher Art



Zeitzerlegung mit Bitumkehrordnung am Ausgang

Bild 6.41

Setzt man nun gemäß der Knotengleichungen die Abtastwerte $x_0 \ldots x_7$ ein, ergibt das Vektorelement $\bar{A}_3$

$$\bar{A}_3 = x_0W^0 + x_1W^3 - x_2W^2 - x_3W^5 - x_4W^0 - x_5W^3 + x_6W^2 + x_7W^5.$$

Dieses Polynom steht für die ausgeschriebene Zeile des Produkts aus dem Spaltenvektor der Abtastwerte x_n und der vierten Zeile der W-Matrix des FFT-Butterfly nach Bild 6.40. Der Ausdruck zeigt die vollständige, allgemeine Schreibweise für den Spektralwert $\bar{A}_3$. Er läßt sich noch weiter vereinfachen. Für N = 8 ist

$$W_8 = e^{j\pi/4} = (1 - j)/\sqrt{2};\ j = \sqrt{-1}$$

Setzt man für alle W den Ausdruck $(1 - j)/\sqrt{2}$ ein, ergibt das die Endform der vierten Zeile der W-Matrix. Nach kleinen mathematischen Umformungen präsentiert sie sich in folgender Form sehr anwenderfreundlich:

4. Zeile 1 −jW j W −1 jW −j −W

Leitet man nach dem Überlagerungssatz für jedes Vektorelement $\bar{A}_m$ den vollständigen Ausdruck her und vereinfacht ihn nach obigem Beispiel, erhält man die 8 x 8 W-Matrix für den FFT-Algorithmus. Diese Matrix hat einen außerordentlichen Zuschnitt für eine Berechnung mit dem Computer. Diese Matrix für das FFT-Butterfly in Bild 6.40 sieht folgendermaßen aus:

$$W_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W & -j & -jW & -1 & -W & j & jW \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -jW & j & W & -1 & jW & -j & -W \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -W & -j & jW & -1 & W & j & -jW \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & jW & j & -W & -1 & -jW & -j & W \end{bmatrix}$$

Diese Matrix ist der Kernpunkt einer sehr schnell ausführbaren Fourier-Transformation mit 8 Abtastwerten. Trennt man, gemäß nach Bild 6.39, die Abtastwerte in Real- und Imaginärteil $x_n = a_n \pm jb_n$, ergeben sich einfache Polynome, die (für W) mit dem Faktor $1/\sqrt{2}$ zu multiplizieren sind. In den anderen Fällen muß nur addiert oder subtrahiert werden. In der Matrix sind jeweils die Real- und Imaginärteile der Koeffizienten (mit W und jW in einer Zeile bezeichnet) folgend zu behandeln:

W	−jW	−W	+jW			−jW	W	jW	−W
$(a_1$	$- a_3$	$- a_5$	$+ a_7)$	−	Real	−	$(-a_1$	$+ a_3$	$+ a_5 - a_7)$
$(b_1$	$- b_3$	$- b_5$	$+ b_7)$	−	Imaginär	−	$(-b_1$	$+ b_3$	$+ b_5 - b_7)$

Ferner kann die Berechnung weiter vereinfacht werden, wenn die zu analysierende Funktion Eigenschaften besitzt, wie sie in Bild 6.36 angedeutet wurden. Ist f(t) eine gerade Funktion, ist nur der Realteil von $\overline{A}_m$ zu berechnen. Für eine ungerade Funktion muß nur der Imaginärteil von $\overline{A}_m$ betrachtet werden.

Die hier gezeigten theoretischen Aspekte über die FFT sind als das absolute Mindestmaß anzusehen, das man wissen sollte, um eine Signalanalyse mit dem Computer durchzuführen.

Die programmtechnische Ausführung der Matrix-Multiplikation mit dem Z 8000 wurde bereits prinzipiell dargestellt. Das weitere Programm in Assembler besteht aus elementaren Operationen.

Bild 6.43 zeigt schließlich die praktische Anordnung eines Z 8000-Systems, mit dem sich eine Signalanalyse durchführen läßt. Es sei nochmals auf Kapitel 4 verwiesen, in dem man die spezifische Hardware vorfindet und zusammenstellen kann. Im folgendn Abschnitt 6.5 wird auf die Hardware eines A/D-Interfaces eingegangen.

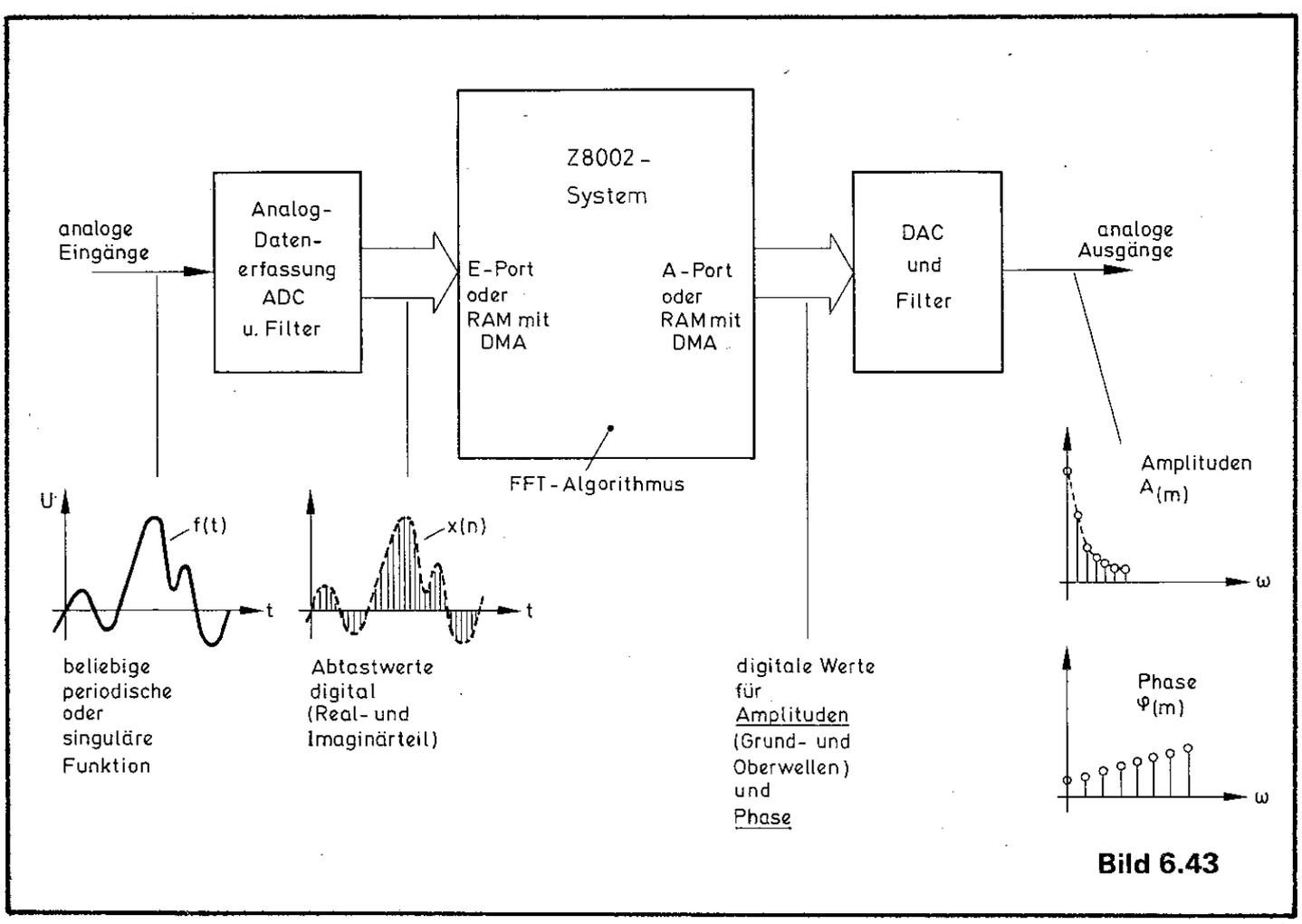


Bild 6.43

6.5 BEISPIEL EINER ANALOGEN DATENERFASSUNG MIT EINEM 16-BIT-ANALOG-DIGITAL-UMSETZER

Ein Computer lebt gleichsam durch vielfältigen Datenaustausch. Ein Prozeßrechner kann nur dann seiner spezifischen Aufgabe nachkommen, wenn er on-line mit den einzelnen Prozeßoperationen verbunden ist. Das Interface für die Übernahme von Zustandsvariablen, insbesondere in Form analoger elektrischer Meßwerte ist der Analog-Digital-Umsetzer, dessen unterschiedliche Funk-

tionsprinzipien in vielerlei Typen Anwendung finden. Im folgenden Hardwarebeispiel wird einer der extremsten Umsetzer für ein Interface zum Z 8000 vorgestellt. Es handelt sich um den hybriden 16-bit-Analog-Digital-Umsetzer der Firma Burr-Brown. Extrem ist dieser ADC schon allein deshalb, weil er ein analoges Spannungssignal in ein digitales 16-bit-Wort verschlüsselt. Diese hohe Auflösung des als Stufenverschlüssler arbeitenden ADC bewirkt, daß ein Eingangssignal im Bereich 0 . . . +10 V im µV-Bereich aufgelöst wird. Er benötigt eine Umsetzzeit von 57 µs für 16 bit und hat bei einer Umgebungstemperatur von 25 °C (nach einer Anwärmzeit von 5 Minuten) eine minimale Abweichung von ± 0,003 % bezüglich der idealen Übertragungs-Kennlinie (betrifft ADC 71 KG). Das interne R-2R-Dünnfilm-Widerstandsnetzwerk ist mit Hilfe eines Laserstrahls auf das Verhältnis 1:2 sehr genau abgeglichen worden. **Bild 6.44** zeigt das Anschlußbild dieses, in der Tat extremen Bausteins.

Der Anschluß 31 bezeichnet einen Eingang zur Auslösung einer Umsetzung. Der Start erfolgt bei einer angelegten Hi↓Lo-Flanke. Der Eingang muß für die Dauer der Umsetzung auf Lo bleiben. Der Anschluß 18 des ADC bezeichnet einen Ausgang, der eine Information über den Status der Umsetzung liefert. Der Ausgang zeigt während der Umsetzung Hi an und liefert mit einer Änderung von Hi auf Lo die Anzeige, daß sich der niederwertigste Digital-Ausgang (LSB) des ADC auf einen Wert stabilisiert hat. Die Umsetzung ist fertig. Bleibt der Start-Eingang 31 auf Lo, verharrt der ADC in einem stationären Zustand, d.h. die Digital-Ausgänge bleiben unverändert. Dies ist auch weiterhin der Fall, wenn der Eingang von Lo auf Hi gesetzt wird. Ein neuer Umsetzzyklus beginnt effektiv erst bei einer Hi↓Lo-Flanke an diesem Eingang. Dieser asynchrone Steuervorgang ist im Signaldiagramm in **Bild 6.45** zu ersehen. Vor der Startflanke muß der Start-Eingang mindestens 50 ns auf Hi gewesen sein.

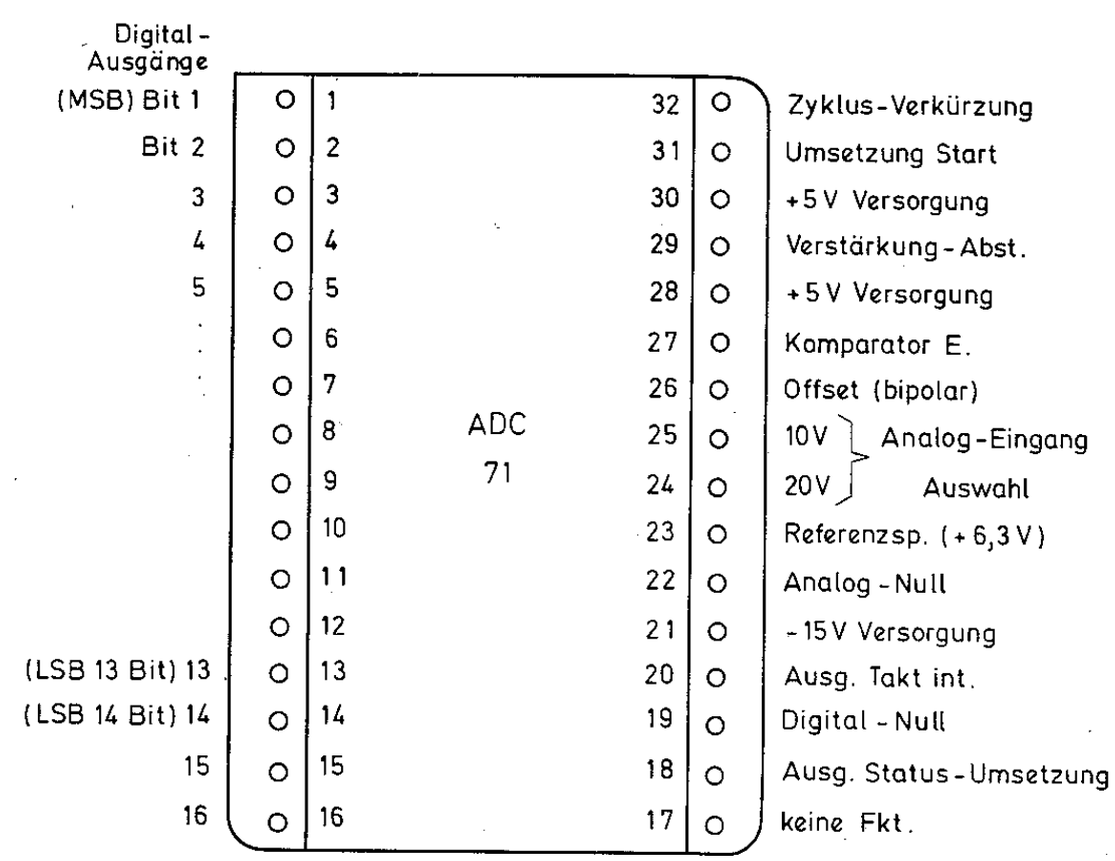


Bild 6.44

Die Digital-Ausgänge des ADC sind in **komplementärer Logik** ausgeführt. Im Gegensatz zu Bild 6.45 gelten die Quantisierungsstufen bei der Angabe "1" für den Pegel Lo. Als Eingangsspannungsbereich für das Analogsignal sind die bipolaren Bereiche ± 10 V, ± 5 V und ± 2,5 V und die unipolaren Bereiche 0 . . . +5 V, 0 . . . +10 V und 0 . . . +20 V zulässig. Wird der Analog-Eingang mit +10 V max. angesteuert, liefern die Digital-Ausgänge Lo-Pegel; für 0 Volt werden folglich Hi-Pegel ausgegeben. Für die bipolaren Bereiche wird für ± 0 Volt der digitale Wert 0111 . . . 11 ausgegeben. Als Referenz für die Quantisierung sei hier angegeben, daß im Bereich 0 . . . +10 V das LSB bei einer 14-bit Auflösung eine Schwelle von 610 μV besitzt, bei der eine Änderung von logisch 0 auf 1 eintritt.

Die Interface-Schaltung mit dem ADC 71 besteht aus zwei Teilen.
Bild 6.46 zeigt einen Teil. Über einen nicht invertierenden Spannungsfolger (LM 302) wird das Analogsignal eines Bereichs von ± 10 V an den Eingang 24 des ADC gegeben. An den beiden 100-kΩ-Trimmern wird der Offset und die Skalierung eingestellt. Die Digital-Ausgänge werden an zwei invertierende 8-bit-Zwischenspeicher AmZ 8133 angelegt. Damit ist der Komplement-Ausgang des ADC korrigiert. Für den Start des Umsetzvorgangs wurde ein besonderer Auslösemechanismus gewählt. Das Startsignal wird unmittelbar von dem Multi-Mikro-Ausgang $\overline{\mu 0}$ der Z 8000-CPU ausgesandt. Ändert $\overline{\mu 0}$ seinen Zustand von Hi auf Lo ($\overline{\mu 0}$ ist aktiv), wird durch die Gatterverknüpfung vor 31 des ADC ein Startimpuls erzeugt, der, wie

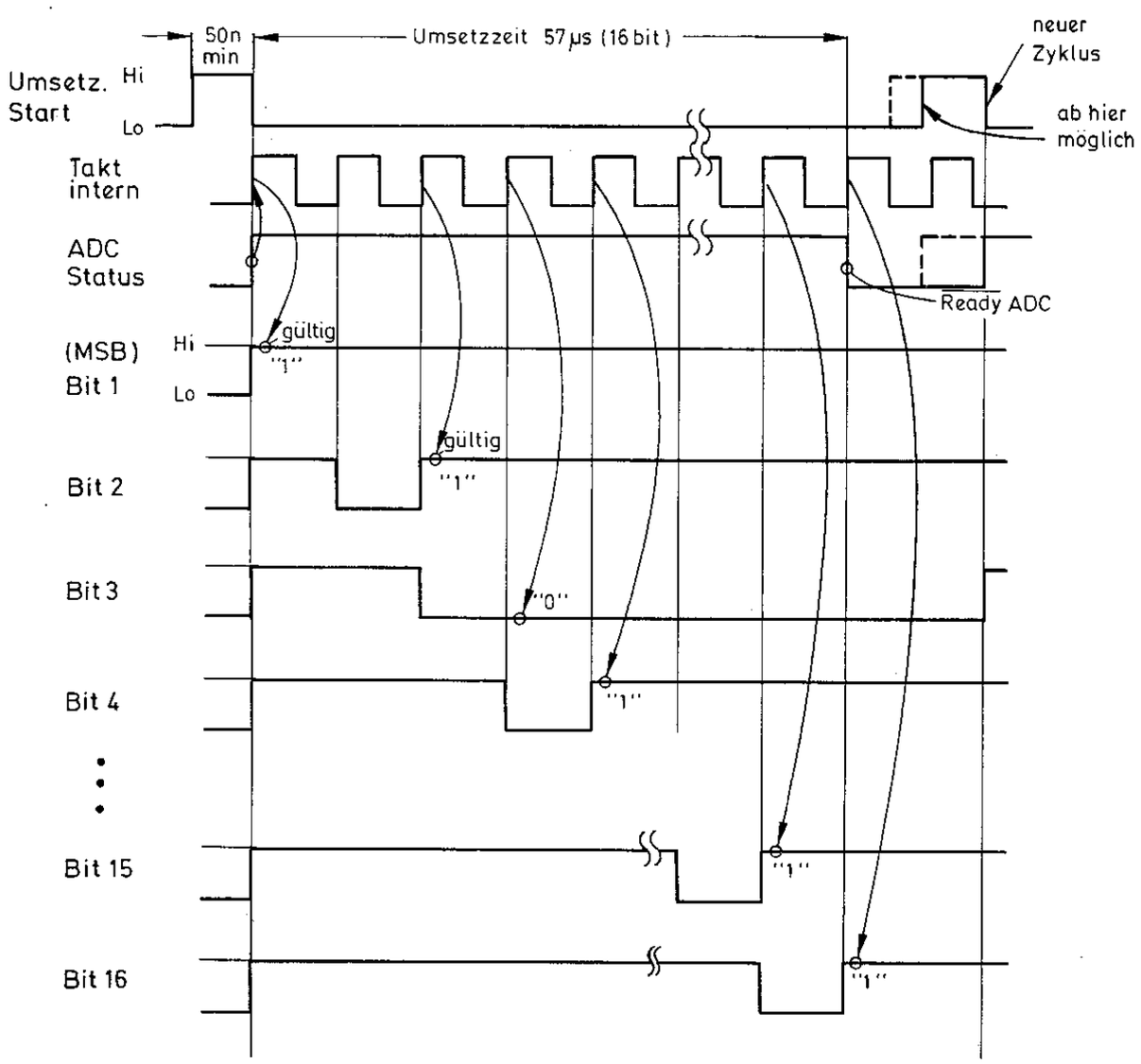


Bild 6.45

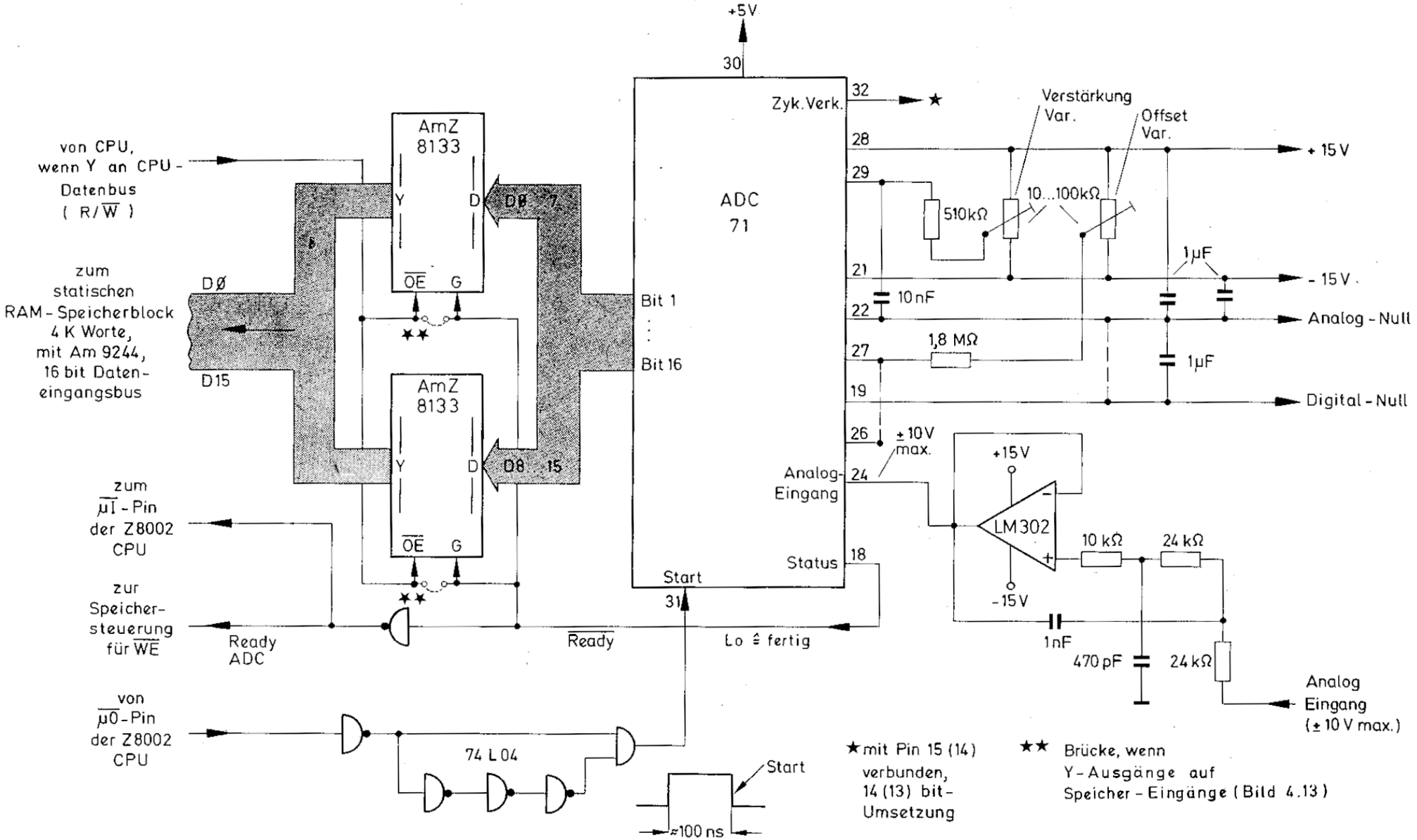

+5V
30
ADC
71
Zyk.Verk.
32
Verstärkung
Var.
Offset
Var.
28
+ 15 V
29
510kΩ
10...100kΩ
1µF
21
- 15 V
22
10nF
Analog - Null
27
1,8 MΩ
1µF
19
Digital - Null
26
± 10V
max.
Analog-
Eingang
24
Bit 1
Bit 16
Status
18
Start
31
AmZ
8133
Y
D
DØ 7
OE
G
AmZ
8133
D8 15
von CPU,
wenn Y an CPU -
Datenbus
(R/W)
zum
statischen
RAM - Speicherblock
4 K Worte,
mit Am 9244,
16 bit Daten-
eingangsbus
DØ
D15
zum
µI - Pin
der Z8002
CPU
zur
Speicher-
steuerung
für WE
Ready
ADC
Ready
Lo ≙ fertig
von
µ0 - Pin
der Z8002
CPU
74 L 04
Start
≈100 ns
+15V
LM302
-15V
10 kΩ
24 kΩ
1nF
470 pF
24 kΩ
Analog
Eingang
(± 10 V max.)
★ mit Pin 15 (14)
verbunden,
14 (13) bit -
Umsetzung
★★ Brücke, wenn
Y - Ausgänge auf
Speicher - Eingänge (Bild 4.13)

Bild 6.46

dargestellt, etwa 100 ns breit ist. An seiner Hi↓Lo-Flanke beginnt die Umsetzung. Die Fertigmeldung $\overline{\text{Ready}}$ gelangt zuerst an G der beiden Latches, wodurch das ADC-Ausgangswort zwischengespeichert wird. Die Latch-Ausgänge liegen am allgemeinen Datenbus an, so daß ein ADC-Wert direkt von der CPU übernommen werden kann. Das wäre üblich. Darüber hinaus kann man sich jedoch des Speicherwerks in Bild 4.13 (Kapitel 4) bedienen.

Die Ausgänge D0 . . . D15 der beiden 8133 werden statt auf den CPU-Bus auf den 16-bit-Dateneingangs-Bus der Speicherbausteine 9244 gelegt. Der Status-Ausgang des ADC wird invertiert und zur Speichersteuerung (Bild 4.13) geleitet. Sobald eine Umsetzung beendet ist, kann über Ready das Einschreiben des ADC-Wertes an eine Speicherstelle des 4K-Worte-Speicherwerks erfolgen. Die CPU führt hierfür einen Pseudo-Schreibzyklus aus, der nur mit einer bestimmten Adresse versehen ist. Die CPU sendet irgendwelche Daten aus, die sich aber auf dem Bus verlieren. Die echten Daten werden ohne Störung unmittelbar von den Latches in das Speicherwerk übernommen. Zuständig hierfür ist das UND-Gatter in Bild 4.13. Die Fertigmeldung Hi liegt am UND an. Über $R/\overline{W}$ gleich Lo der CPU ist der UND-Ausgang Lo und somit an $\overline{WE}$ der Speicher Lo. Für Ready gleich Lo ist am UND-Ausgang immer Lo, was einfach besagt, daß die CPU während eines Umsetzvorgangs nicht von diesem Speicherwerk auslesen kann. Die Information über die Beendigung des Umsetzvorgangs bekommt die CPU am $\overline{\mu I}$-Eingang mitgeteilt, d.h. ist $\overline{\text{Ready}}$ gleich Hi, ist $\overline{\mu I}$ damit Hi und zugleich inaktiv. Bei der Anwendung des MBIT-Befehls der CPU wird das S-Flag folglich nicht gesetzt. $\overline{\mu 0}$ kann auf Grund dessen neu aktiviert werden (mit dem MSET-Befehl). Es ist hier in jedem Fall von Vorteil, wenn diese Bedingung noch durch ein NOR-Gatter ausgebaut wird. In **Bild 6.47** ist der triviale Zusammenhang erläutert. Der zweite Eingang wird als Ereignis-Eingang benützt. Ist an diesem Eingang ein Lo, wird, wenn an Ready ein Lo liegt, der $\overline{\mu I}$-Eingang gleich Hi. Damit wird das gleiche erreicht wie zuvor. $\overline{\mu I}$ ist inaktiv (S-Flag gleich 0), so daß durch die Anwendung des MREQ-Befehls der $\overline{\mu I}$-Eingang und folglich $\overline{\mu 0}$ auf Lo (aktivieren) gesetzt wird. Damit wird der ADC gestartet.

Mit der gezeigten Hardware und den Z 8000-Multi-Mikro-Befehlen ist damit eine Analog-Digital-Umsetzung unter der Ereignisbedingung $\overline{\text{Ereignis}}$ gleich Lo programmierbar. Die Auslösung der Abfrageschleife für $\overline{\mu I}$ kann üblicherweise mit einem Interrupt durch die Ereignisleitung erfolgen. Das muß jedoch nicht sein. Die Abfrageschleife kann man beliebig in ein Programm einbauen. Folgende kleine Routine betrifft die Übernahme eines Analogwertes (nach Bedingung) mit der vorgegebenen Logik des Interfaces:

Zeile	Mnemonic	Bezeichnung	Bemerkung
1	LDA	R 10 , ADC-SPEICHER	! Startadresse für Speicher !
2	LD	R 0 , #31	! Verzögerung 57 µs, Umsetzung !
3	MREQ	R 0	! $\overline{\mu 0}$ gesetzt, Laufzeit, $\overline{\mu 0}$ rückgesetzt !
4	TEST	R 0	! war R 0 gleich 0 ? !
5	JR	NZ , MREQ	! ADC-Start wiederholen, wenn Z = 0 !
6	LD	@ R 10 , R 0	! ADC-Wert von Latch in Speicher !
7	INC	R 10 , # 2	! nächste Adresse für Speicher !

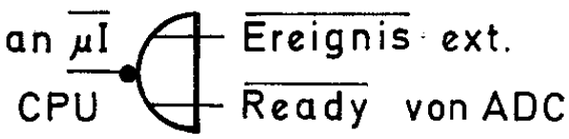


Bild 6.47

Mit diesem kleinen Programm wird ein analoger Wert mit dem ADC digitalisiert und an eine vorbestimmte Speicherstelle des Speicherwerks in Bild 4.13 transferiert. Man kann eine Abfrage anhängen, bis zu welcher Adresse der Speicher geladen werden soll. Es sei zu MREQ noch bemerkt, daß die Verzögerung mit #31 in R 0 so bemessen ist, daß $\overline{\mu I}$ unbedingt inaktiv sein wird, so daß $\overline{\mu 0}$ nach dem Durchlauf der Verzögerung am Ende des Befehls ebenfalls wieder inaktiv wird. MREQ startet den ADC und bringt auch zuletzt alles wieder zur Ruhe.

Ein Multiprozessor-Konzept mit Z8000-Bausteinen

7

Eine logische, wenn auch triviale, Überlegung bei der Bearbeitung genereller programmtechnischer Aufgaben (Prozesse) ist, eine interne Koordinierung eines im allgemeinen sequentiellen Prozesses auszuarbeiten. Ein Prozeß ist eine sequentielle Folge von Einzelaktionen. Ist der Prozeß bezüglich dieser Aktionen nicht ausschließlich an eine starre, sequentielle Folge gebunden, ist eine Parallelisierbarkeit bei der Bearbeitung von Einzelaktionen zumindest im Bereich des möglichen. Eine Analyse und Koordienierung des Prozesses soll dies herausfinden. Ein Prozeß besteht, genaugenommen, aus einer Anzahl voneinander unabhängiger Einzelaktionen im Fluß der sequentiell abhängigen. Ergibt die Analyse eines Prozesses eine endliche Zahl unabhängiger Einzelaktionen in ein erwägbares Verhältnis zu den abhängigen, muß eine Parallelbearbeitung dieser Einzelaktionen theoretisch einen Zeitvorteil bringen. Betrachtet man üblicherweise einen Rechner mit einer CPU als Monosystem, ist die Durchführung eines Prozesses nach der bekannten Flußrichtung rein sequentiell. Sobald mehr als eine CPU an der Bearbeitung eines Prozesses beteiligt ist, kann praktisch ein Zeitvorteil gegenüber dem Monosystem erzielt werden. Damit ist die triviale Überlegung in ihrer Zielrichtung plausibilisiert.

Die Bedingung an eine Parallelbearbeitung von Einzelaktionen zum Ziel einer schnelleren Ausführung eines Prozesses sind jedoch sehr vielfältig. Man darf sich hier nicht täuschen lassen. In Kapitel 2, Abschnitt 11, wurde bereits von einer Parallelisierbarkeit eines Prozesses ausgegangen. Die Beschreibung betraf bereits Feinheiten, wie eine Ausführung unter Beteiligung mehrerer Prozessoren anzugehen ist. Es bedarf in jedem Fall erheblicher Überlegungen, aus der Analyse eines komplexen Prozesses eine faktisch nachweisbare Zeitersparnis zu errechnen, die den Aufwand einer Parallelisierun lohnt, zumal eine der Unbekannten in der Hardware des Parallelnetzwerkes von CPUs verborgen liegt. Überdies darf man sich keinesfalls der Ansicht hingeben, daß zwei Prozessoren die gegebene Anzahl von Einzelaktionen in der Hälfte der Zeit bearbeiten. Ein Prozeß, der von kooperierenden Prozessoren bearbeitet wird, muß vor und während der parallelen Ausführung der Aktionen koordiniert sein, was eine endliche, nicht zu vernachlässigende Zeit erfordert. Parallele Aktionen haben einen definierten Anfang und ein präzises oder relativ definiertes Ende. Einzelne Aktionen entscheiden über Anfang und Ende abhängiger Aktionen. Eine völlig willkürliche Zuordnung relativ unabhängiger Einzelaktionen eines Prozesses gibt es nicht. Ein gewisser Rahmen eines logischen Zusammenhangs aller an einem Prozeß beteiligten unabhängigen Einzelaktionen wird immer gegeben sein.

Die Freiheitsgrade einer zeitlichen Verschiebung und damit der Koordinierbarkeit ist selbstverständlich von Prozeß zu Prozeß unterschiedlich. Dies bedarf eben der schon erwähnten, genauen Analyse. Die genaue Analyse der Zuordnung abhängiger Einzelaktionen zu den unabhängigen eröffnet erst einmal eine theoretische Optimierung der Ausführungszeit des Prozesses, ohne daß das Parallelrechner-Netzwerk in irgendeiner Form bekannt sein muß. Den einzigen Parameter, den die theoretische

Analyse genau zu fixieren gestattet, ist die Zahl der an einem Prozeß beteiligten Prozessoren. Was bleibt also? Hat man den Prozeß soweit "im Griff", folgt die Überlegung der koordinierten Zusammenarbeit zwischen den Prozessoren untereinander. Der Freiheitsgrad einer Koordinierung zielt nun schon etwas genauer auf das Rechnernetzwerk ab. Hier gibt es nun allerlei Möglichkeiten einer Zusammenarbeit, die sich auf gleicher wie priorisierter Ebene abspielt. Der Rang einer CPU und damit die Kompetenz eines Zugriffs ist von dem Freiheitsgrad abhängig, der in der Bus-Hardware implementiert ist. Hardwaretechnisch äußerst sich dies in der Multibus-Konfiguration, die die diversen Querverbindungen unter den CPUs und zum Haupt-Resource erlauben. Die scheinbar klassische Struktur (sofern man überhaupt davon sprechen kann) ist die Zusammenschaltung einer Anzahl von Prozessoren auf einen gemeinsamen Adreß-, Daten- und Steuerbus, der über eine Verkettung zu einer Busrequest-Staffel für jeden einzelnen Prozeesor einen Zugriff auf das Haupt-Resource (das Arbeitsspeicherwerk) gestattet. Damit ist eine Hardwarestruktur angerissen, die bereits vereinzelt in der einschlägigen Literatur behandelt wurde. Es sei hier nochmal an Bild 2.53 in Kapitel 2 erinnert, das eine priorisierte Kette zeigt. Angefügt sei jedoch, daß mit der Struktur: mehrere Prozessoren auf eine Quelle, noch nicht viel gelöst ist. Die Probleme bei der Konzeption eines effektiven Parallelprozessor-Netzwerks sind viel zu kompliziert, als daß mit der genannten Methode der einzig wahre Weg vorgeschrieben ist. Jeder Zugriff einer CPU auf das Resource bedarf einer Anforderung, die erfüllt sein will. Um so mehr Prozessoren in der Kette verbunden sind, um so langwieriger ist der Anerkennungsprozeß (im Maximalfall) für einen einzigen Zugriff einer CPU. Damit ist die Problematik bei Parallelrechnern nur ein wenig angedeutet. Verwendet man zunehmend mehr Prozessoren in einem Netzwerk, muß das dem eigentlichen Ziel, bei der Ausführung eines Prozessors Zeit zu sparen, nicht unbedingt dienlich sein. Rechnernetzwerke der Art nach Bild 2.53 können zeitlich ziemlich unwirtschaftlich arbeiten, wenn man die Koordination ungeschickt ausarbeitet.
Für die Konzeption von Parallelrechner-Netzwerken seien einige Grundsätze genannt, die es zu beachten gilt.

- Anzahl unabhängiger Einzelaktionen eines Prozessors in Relation zur Anzahl verwendeter Prozessoren bringen
- Anzahl der verwendeten Prozessoren zur Anzahl unabhängiger Einzelaktionen niemals größer 1 wählen, eher kleiner; Optimierung erforderlich
- Genaue Auflistung erforderlicher Zugriffe einer CPU auf das Haupt-Resource während der Ausführung der Einzelaktionen erstellen
- Prioritätsschema einer Einzelaktion mit Gewichtung auf Anzahl der Zugriffe und Rangordnung im Prozeß erstellen
- Aufgrund des vorhergehenden Punktes grundsätzliche Überlegungen über die Autonomie eines Prozessors inn der Kette anstellen.
  Frage: Ist der Ausbau eines Prozessors zu einem abgeschlossenen Subsystem von Vorteil, um damit die Anzahl der Zugriffe auf das Haupt-Resource zu minimieren?

Die Zahl elementarer Zugriffe auf das Haupt-Resource (Bedarf des Multibusses) muß minimiert werden, um eine ohnehin entstehende Warteschlange von Anforderungen so klein wie nur irgend möglich zu halten. Eine Warteschlange bringt für jeden Prozessor eine Totzeit bei der Ausführung einer Aktion. Die Totzeit bringt eine Verlängerung der Rechenzeit, die man mit einem Parallelrechner gerade zu umgehen wünscht.

7.1 PARALLELRECHNERSYSTEME ALLGEMEIN

Die bereits vielfach genannte Einzelaktion eines Prozesses wird im folgenden etwas genauer betrachtet. Man stelle sich als auszuführende Aufgabe (den Prozeß) den simplen Fall einer 16-bit-Addition vor. Die Ausführung soll mit einem 4-bit-Slice-Prozessor erfolgen, beispielsweise mit dem Standardtyp A, 2901A. Es sollen die beiden Werte (1) = 41404 und (2) = 52901 addiert werden. Numerisch ergibt das:

(1)		1	0	1	0	0	0	0	1	1	0	1	1	1	1	0	0	= A1BCH
(2)		1	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1	= CEA5H
	1				1				1				1					← Überträge
Summe	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	= 17061H

Stelle man in einfacher Form den Aktionsfluß des Prozesses (besser Prozedur) zusammen, ergibt sich abgekürzt folgender Verlauf:

Zeile 1 Eingabe Operanden 2^0 . . . 2^3: 1100, 0101
Zeile 2 erste Addition
Zeile 3 Ergebnis (0001) abspeichern
Zeile 4 Carry Cn+4 retten
Zeile 5 Eingabe Operanden 2^4 . . . 2^7: 1011, 1010
Zeile 6 Carry von 2 anlegen
Zeile 7 zweite Addition
.
.
.
letzte Zeile: Ergebnis 10111 abspeichern.

Der triviale Ablauf erfordert keinen weiteren Kommentar. Bei dem Beispiel ist sicher einleuchtend, daß die Reihenfolge der Prozedur einzuhalten ist. Es handelt sich hier um voneinander abhängige Einzelaktionen. Will man nun diesen sehr einfachen Fall parallelisieren, ist hier unschwer erkennbar, daß durch die Zusammenschaltung von vier 2901A-Prozessoren die vier Additionen als Einzelaktion parallelisierbar sind. Die Additionen sind jedoch noch nicht als unabhängige Aktionen anzusehen, weil jede Addition ein Carry-Bit erzeugt, das für die folgende zu verwerten ist. Die Prozedur ist parallelisierbar, wenn in diesem Fall die hardwaremäßige Unterstützung durch den Carry-Look-Ahead-Baustein Am 2902A gegeben wird. Dieser Baustein addiert alle auftretenden Überträge parallel. Die Addition und damit die Prozedur basieren jetzt auf unabhängigen Einzelaktionen.

Es muß zu diesem Beispiel vorweg angefügt werden, daß einkaskadiertes Bit-Slice-System mit 29er-Bausteinen kein Multiprozessorsystem ist, wie es im Ver-

lauf dieses Kapitels noch vorgestellt wird. Das Beispiel zeigt in seiner elementaren Art das Prinzip einer Parallelisierung. Es zeigt die abhängigen Einzelaktionen des 4-bit-Rechners in eine Umsetzung unabhängiger Einzelaktionen. Dieses Beispiel ist in seiner Art auf 8-bit-CPUs und auch auf den Z 8000 prozezierbar.

Die Aufgaben an einen Parallelrechner sind kompliziert und meist vielschichtig, zeigen jedoch generell zwei gemeinsame Eigenschaften:

- überwiegend rechenintensive, numerische Prozeduren,
- hohe Anforderung bezüglich kurzer Ausführungszeiten.

Es gibt eine Vielzahl programmtechnischer Aufgaben, die im wahrsten Sinne des Wortes viel Routine enthalten, d.h. die repetierende Berechnung mathematischer Ausdrücke, die sich letztlich in Form einer umfangreichen Tabelle und einer Punktgrafik im Ergebnis äußert. Ein Monoprozessorsystem berechnet Ausdruck für Ausdruck sequentiell und stellt anschließend die Ergebnisse zusammen. Ein Multiprozessorsystem kann hier im einzelnen beträchtliche Zeit durch die parallele Bearbeitung solcher Ausdrücke gewinnen. Wie können Aufgaben für einen Parallelrechner aussehen? **Bild 7.1** zeigt ein allgemeines Beispiel. Es soll ein umfangreiches lineares Gleichungssystem berechnet werden. Zur Berechnung der Unbekannten kann sowohl eine Matrix-Matrix- wie (überwiegend) eine Matrix-Vektor-Multiplikation notwendig sein. Für diese häufig auftretende Prozedur ist ein Unterprogramm (UPRO) vorgesehen. Der zahlreiche Aufruf des UPROs für die Spalten-Zeilen-Multiplikation ermöglicht in diesem Punkt die Aufsplittung der gesamten Aufgabe auf mehrere Prozessoren. Ein technisches Beispiel hierzu wäre die Berechnung eines umfangreichen elektrischen Netzwerks nach den Kirchhoffschen Knotenpunkt- und Maschenstromgleichungen, die sich bekanntlich in der Matrixform schreiben lassen (erstes Kirchhoffsches Gesetz):

$$\begin{bmatrix} m_{11} & m_{12} & \dots & m_{16} \\ m_{21} & m_{22} & \dots & . \\ . & & . & . \\ . & & . & . \\ m_{j1} & & . & \\ . & & & \\ . & & & \\ m_{k1} & \dots & & m_{kl} \end{bmatrix} \cdot \begin{bmatrix} I_1 \\ I_2 \\ . \\ . \\ . \\ \\ \\ I_l \end{bmatrix} = 0$$

$I_1 \dots I_L$ bezeichnet die Zweigströme. Eine noch allgemeinere Form stellt das zweite Kirchhoffsche Gesetz dar, das in der Vektorform

$$(Z)\overline{I} + (M')\overline{U} = \overline{E}$$

lautet. Z ist die Impedanz-Matrix, I ist der Vektor der Zweigströme, M' ist die transponierte Matrix der Hilfskoeffizienten und U ist der Vektor der Knotenpotentiale. In der Matrixform wird die obige Gleichung folgendermaßen geschrieben:

$$\begin{bmatrix} Z_{11} & Z_{12} & \dots & Z_{12} \\ Z_{21} & Z_{22} & . & . \\ . & & . & . \\ . & & . & . \\ . & & & \\ Z_{l1} & Z_{2l} & \dots & Z \end{bmatrix} \cdot \begin{bmatrix} I_1 \\ I_2 \\ \\ \\ I_l \end{bmatrix} + \begin{bmatrix} m_{11} & m_{21} & \cdots & m_{k1} \\ m_{12} & m_{22} & & m_{k2} \\ . & & & \\ . & & , & \\ . & & & \\ m_{1l} & m_{2l} & \dots & m_{kl} \end{bmatrix} \cdot \begin{bmatrix} U_1 \\ U_2 \\ . \\ . \\ . \\ U_k \end{bmatrix} = \begin{matrix} \text{Zweig-} \\ \text{ursprung} \end{matrix} \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ . \\ E_l \end{bmatrix}$$

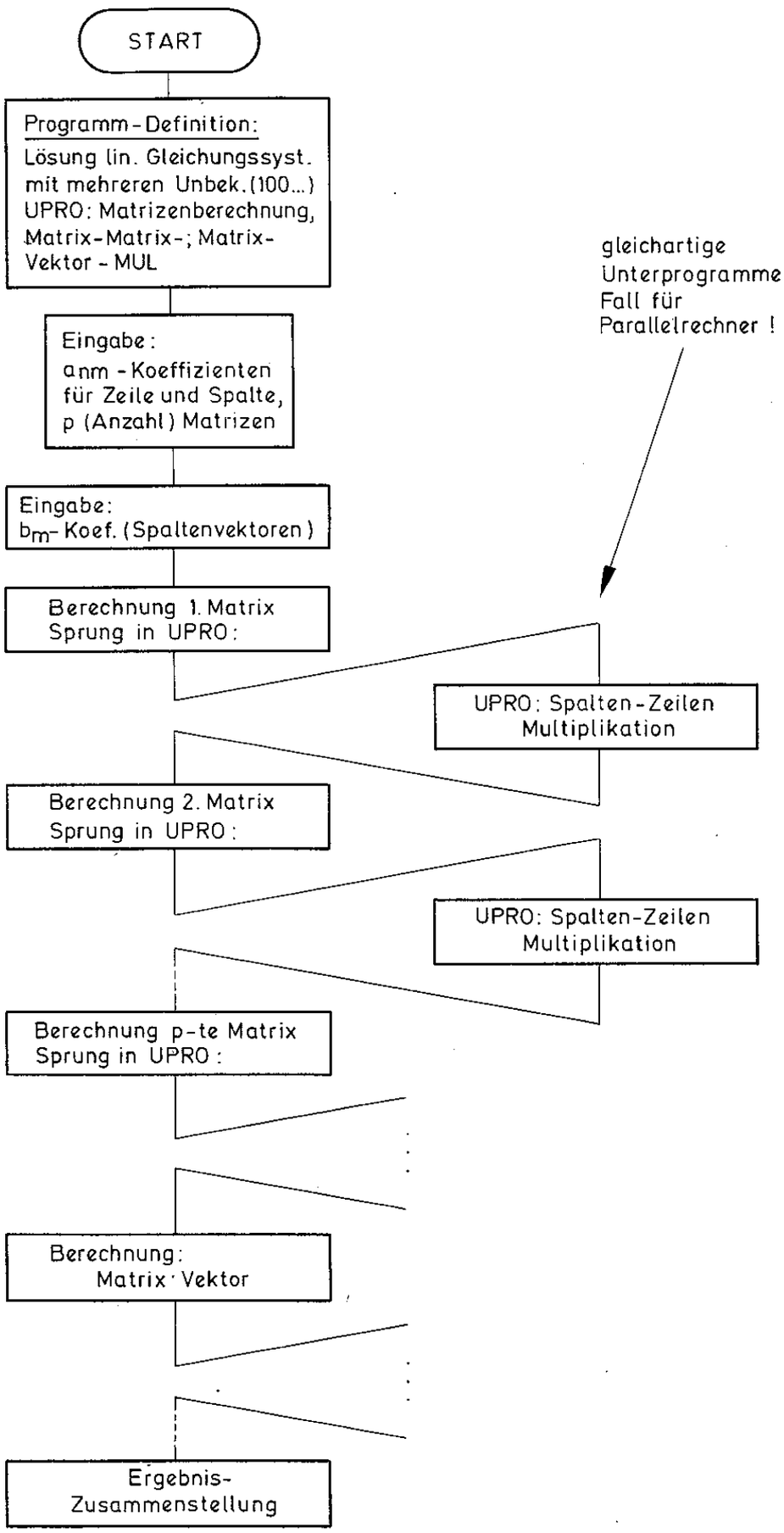

START
Programm-Definition:
Lösung lin. Gleichungssyst. mit mehreren Unbek.(100...)
UPRO: Matrizenberechnung, Matrix-Matrix-; Matrix-Vektor - MUL
Eingabe:
anm - Koeffizienten für Zeile und Spalte, p (Anzahl) Matrizen
Eingabe:
bm- Koef. (Spaltenvektoren)
Berechnung 1. Matrix
Sprung in UPRO:
gleichartige Unterprogramme Fall für Parallelrechner !
UPRO: Spalten-Zeilen Multiplikation
Berechnung 2. Matrix
Sprung in UPRO:
UPRO: Spalten-Zeilen Multiplikation
Berechnung p-te Matrix
Sprung in UPRO:
Berechnung:
Matrix · Vektor
Ergebnis-Zusammenstellung

Bild 7.1

Die Berechnung der Einzelglieder erfolgt, wie schon genannt, aus der Multiplikation von Zeile mal Spalte: $Z_{11} \cdot I_1 + Z_{12} \cdot I_2 \ldots$, ebenso bei der Matrix M' mit den Knotenpotentialen. Bei einem entsprechend großen Netzwerk kann die Berechnung der Matrizen maßgebende Zeit beanspruchen. Sind beispielsweise 100 Zweigströme und 60 Knotenpotentiale zu berechnen, könnten die jeweilig notwendigen Zeilen/Spalten-Routinen auf 4 Prozessoren verteilt werden. Es hängt sowohl von der Leistungsfähigkeit der Prozessoren wie auch von der Organisationssoftware ab, wie eine effektive Teil-Task-Verteilung zu vergeben ist. Ohne ein genauer definiertes Parallelrechner-System vorgegeben zu haben, ist eine Angabe der Verteilung nur unzulänglich zu geben.

Ein wichtiger Punkt ist, wie der Parallelrechner in seiner Hardwarestruktur aussehen kann. Welche Kompetenz darf ein Subprozessor haben. Es gilt, wie eingangs von Kapitel 7 erläutert, die Benützung von Multibusleitungen gering zu halten. Besteht das System aus mehreren gleichartigen und gleichberechtigten Prozessoren, ist ein Konflikt bei Zugriffen auf Haupt-Resourcen nicht auszuschließen. Parallele Aktionen können nicht völlig unabhängig voneinander ablaufen. Es muß zwischen ihnen eine minimale Form einer Kommunikation stattfinden, um die Konfliktsituation gleichzeitiger Zugriffe zu vermeiden. Die minimale, aber ausreichende Kommunikation der Prozessoren untereinander geschieht nach dem Verfahren der priorisierten, verketteten Anforderung, wie es in Kapitel 2, Abschnitt 11, dargestellt wurde. Der sogenannte Katastrophenfall eines System-deadlocks (Verklemmung) wird vermieden. Der Grund liegt in der systematischen Organisation durch die "daisy-chain"-Buskonfiguration. Der Zugriff auf den Hauptspeicher wird nur erlaubt, wenn die entsprechende Bedingung erfüllt ist. Von Gleichberechtigung der Prozessoren kann jedoch hier keine Rede mehr sein. Die Anforderung auf einen Zugriff wird in einer Staffel bearbeitet, die sich bei einer entsprechenden Zahl von vorliegenden Anforderungen in eine Warteschlange von Anforderungen ausbildet. Die technische Gegebenheit der Verkettung bedingt eine starre Behandlung der Warteschlange, weil ihr Abbau einzig durch die Priorisierung geregelt ist. Betrachtet man die Gleichberechtigung nur in dem Punkt, daß jeder Prozessor einen Zugriff auf den Hauptspeicher vornehmen darf, wann auch immer er genehmigt wird, ist Gleichberechtigung vorhanden. Würde Gleichberechtigung zugleich als Zugriff betrachtet werden, ohne eine Warteschlange durchlaufen zu müssen, ist eine individuelle Buszuführung vom Prozessor auf das Haupt-Resource und eine Mehrfachadressierung notwendig. Da dies technisch unwirtschaftlich ist, muß zur Reduzierung der Warteschlange ein anderer Weg gegangen werden.

Es wurde der Begriff der Autonomie eines Subprozessors genannt. Genaugenommen bedeutet dies: Ein Subprozessor ist zu einem System mit Arbeitsspeicher und eigener Firmware ausgebaut. Bekommt ein Subprozessor eine Teil-Task, wird er mit allem versorgt, was zur Ausführung notwendig ist, d.h. mit dem entsprechenden Programm und den erforderlichen Daten. In seinem angegliederten Arbeitsspeicher befindet sich gewissermaßen die Aufgabenstellung und was dazu gehört. Der Subprozessor kann die Aufgabe ausführen, ohne einen Bedarf auf den Hauptspeicher anzumelden. Diese Überlegung ist grundlegender Art und wurde bereits erfolgreich bei Versuchen von Parallelrechner-Netzwerken verwendet. Um hier noch konkreter zu werden, das Subsystem könnte auch als Einchip-Mikrocomputer ausgebildet sein. Als Beispiel sei der Z 8 von Zilog genannt, der 2 KB maskierbaren Festwertspeicher und 128 Byte RAM enthält, der extern bis 62 KB erweiterbar ist. Ohne weiter auf den Z 8 einzugehen, ist dies in jedem Fall eine Möglichkeit der Ausbildung eines Subsystems auf kleinstem Raum. Das angestrebte Ziel bei der Überlegung zu mehr

Eigenständigkeit des Subsystems ist die Reduzierung auf Hauptspeicherzugriffe, die mit dem genannten System erreicht wird. Eine größenordnungsmäßig bedeutsame Art eines Parallelrechners mit extensiv ausgebildetem Subsystem wurde in Bild 2.55 angedeutet. Die übergeordnete Organisation erfolgt durch einen Host-Computer, der u.a. für die Initialisierung und für die Strukturierung der zu parallelisierenden Quellprogramme dienen soll. Damit ist die Zielrichtung künftiger Parallelrechner im Format eines Großcomputers aufgezeigt.

7.2 Z 8000 HOST- UND SUBSYSTEM

In diesem Kapitel wird auf eine konkrete Parallelrechner-Konfiguration hingelenkt, die noch relativ überschaubar, aber dennoch einen äußersten Grad an Leistungsfähigkeit bietet. In ihrer prinzipiellen Art ist es in Bild 2.55 strukturiert dargestellt. Das System besteht aus einem kompletten, für sich autonomen Hauptrechner mit Speicher und Schnittstelle. Der Kern des Hauptrechners wird aus einer Z 8001-CPU gebildet, — wie sollte es auch anders sein. Die CPU wird als Host-CPU bezeichnet, weil ihr generell zwei Aufgaben unterstehen. Die erste, in der Firmware (dem Betriebssystem) festgelegten Aufgabe bezieht sich auf den Dialog mit dem Anwender. Das Betriebssystem erlaubt die übliche Korrespondenz über die bekannten Ein-/Ausgabe-Hilfsmittel. In der Firmware ist nicht nur das Betriebssystem, sondern ein Cross-Compiler einer problemorientierten Sprache und, sicherlich wünschenswert wie notwendig, ein Makro-Assembler enthalten. Im speziellen Fall ist hier PASCAL gemeint, weil für den Z 8000 ein optimierter Compiler existiert. Mit Hilfe dieser Unterstützung, Betriebssystem, PASCAL und Makro-Z 8000, ist ein Dialog möglich und prinzipiell im Fall des Monosystems mit üblichen Computern der höheren Leistungsklasse vergleichbar. Man ist in der Lage, ein Programm in PASCAL auszuführen, ohne spezifische Einschränkungen irgendwelcher Art.

Der zweite Teil dieses Rechners besteht aus einem bisher absolut unüblichen Zusatzwerk. Der Host-Computer wird von einer Vielzahl völlig eigenständiger Subsysteme unterstützt, die zu nichts anderem existieren, als dem Host-Computer Routine-Prozeduren vielfachen Bedarfs abzunehmen. Die Subsysteme bilden eine priorisierte Kette mit maximal 128 Systemen in der äußeren Ausbaustufe. Auf die Zahl 128 wird noch eingegangen. Ein Subsystem bildet mit Hilfe eines eigenen Arbeitsspeichers und einer spezifischen Betriebs-Firmware ein eigenständiges Computer-Modul. Alle Subsysteme sind in ihrer Funktion vollkommen dem Host-Rechner untergeordnet. Ihre Steuerung obliegt dem Host-Rechner, wobei es im Grunde nicht viel zu steuern gibt. Der Betrieb dieses Rechners im echten Paralleltask erfordert eine umfangreiche Vorbereitung, die jedoch nicht der Host-Rechner, sondern der Anwender zu leisten hat. Es beginnt damit, aus einer gegebenen PASCAL-Routine zu analysieren, ob sie sinnvoll parallelisierbar ist und, wenn ja, in welche Zahl parallelisierbarer Teiltasks sie aufzubereiten ist. Diese Phase muß als rein theoretisch betrachtet werden. Der Fall liegt anders, wenn von vorneherein eine Routine auf Parallelbetrieb ausgerichtet wird. Die Stelle, an der der Rechner wieder zum Einsatz kommt, liegt in der Simulation eines Teiltasks. In der ersten Simulationsphase wird man einfacherweise noch den Host-Rechner dazu mißbrauchen. Hat man einen Teilalgorithmus der Gesamtaufgabe soweit isoliert und für einen Einzellauf zugeschnitten, werden in der Simulation eine Reihe von Programmeigenwerte ermit-

telt. Dies ist die Rechenzeit für verschiedenartig konfigurierte Datenpermutationen. Es dient einfach dazu, um eine mittlere Rechenzeit herauszufinden, um die Divergenz einer Aktivphase im Parallelbetrieb in Grenzen zu halten. Ein weiterer Punkt ist eine nachträgliche Korrektur von Variablenplätzen und Ergebnisstellen für das Teiltask-Programm im Adreßraum eines Subsystems. Diese Festlegung gehört genaugenommen in die Vorbereitungsphase vor einer Simulation. Da Korrekturen nach den ersten Simulationsläufen noch möglich sind, sei hier darauf verwiesen.

Liegen Rechenzeit und Quell-/Zieladressen für die Variablen und Ergebnisse fest, wird es langsam ernst. Die Aktivphase wird damit eingeleitet, indem die Subsysteme mit dem getesteten Teiltask-Programm geladen werden. Sicherlich wird man nochmal theoretisch abwägen, wie die Priorisierung der Subsysteme die Teiltasks betrifft, es gibt durchaus wichtige, abzuwägende Fälle. Es wird hier jedoch Abstand genommen, da eine genauere Betrachtung vom Thema zu weit ablenkt.

Die Aktivphase, gleichzusetzen mit dem eigentlichen Parallelbetrieb, äußert sich in der Arbeit des Rechners folgendermaßen:

- Alle abhängigen Einzelaktionen einer Prozedur, die unbedingt sequentiell ausgeführt werden müssen, bearbeitet der Host-Computer

- Alle unabhängigen Einzelaktionen (Teiltask-Programme) werden von einem oder mehreren Subsystemen ausgeführt. Die Anzahl der nötigen Subsysteme war Ziel einer vorhergehenden Optimierung.

- Alle Daten-Initialisierungsvorgänge, die die Subsysteme während der Parallelphase betreffen, führt ausschließlich der Host-Computer aus (keine eigene Adressierung des mainframe-RAM vom Subsystem aus vorgesehen).

Der Host-Computer besitzt die übergeordnete Leitung während der Programmausführung. Ein Subsystem kann nur die Bedarfsmeldung von Daten anzeigen. Die Priorität besitzt eindeutig der Host-Computer. Daß dies so ist, liegt ausschließlich in der Konzeption des Parallelrechners. Man kann die Vernetzung des Rechners auch mit einer höheren Zugriffsmöglichkeit der Subsysteme auslegen, wie bereits allgemein angedeutet wurde. Für das in diesem Abschnitt besprochene Konzept soll dies nicht gelten.

Damit ist die zweite Aufgabe des Host-Computers umrissen. Er ist für den Anwender und gleichermaßen für die Initialisierung der Subsysteme zuständig. Er verteilt nach Programm die neuen Aufträge in die jeweilig betreffenden Subsysteme. Die Struktur des Rechners ist somit klar umrissen. Er besteht aus "Master" (gleich Host-Computer) und "Slave" (gleich Subsystem). Der generelle Vorteil einer solchen Parallelrechner-Struktur liegt eindeutig in einer praktisch nicht auftretenden Konfliktsituation bei Zugriffen. Die Anforderung auf einen Bedarf des Host-Computers wird angemeldet und reiht sich in eine Warteschlange ein. Die einzelnen Fertigmeldungen der Teil-Tasks lassen diese Warteschlange entstehen, die in dieser Ausführungsphase unvermeidbar ist. Die Meldungen zeigen dem Host-Computer an, daß die parallele Rechenphase beendet ist (ausschließlich auf die Subsysteme bezogen). Die folgende Datenhol- oder Austauschphase betrifft den Host-Computer. Er holt Daten (Ergebnisse) aus den jeweiligen Subsystemen bzw. ihrem RAM-Betrieb und lagert sie im mainframe-RAM ein. Der Host-Computer kann jetzt mit diesen Daten weiter operieren. Er hat damit eine neue Ausgangsbedingung für den weiteren Programmablauf der Anwenderprozedur sich selbst vorgegeben.

Zusammenfassend wird bezüglich der einzelnen Arbeitsphasen des Rechners erkennbar, daß im wesentlichen vier Phasen auftreten:

Monophase:
Der Host-Computer bearbeitet unabhängig bestimmte Einzel-Tasks der Anwenderprozedur.

Initialisierungsphase:
Der Host-Computer verteilt bestimmte Teil-Tasks auf eine Reihe von Subsystemen (Programme + Daten, oder nur Daten, falls das Programm bereits im Subsystem eingegeben ist).

Parallelphase:
Die Subsysteme führen (unabhängig voneinander) die Teil-Tasks aus.

Datenhol- und Austauschphase:
Der Host-Computer bearbeitet die Warteschlange der Fertigmeldungen in Form eines Transfers vom Sub-RAM in den main-frame-RAM.

Die Einzelheiten werden im folgenden Unterabschnitt behandelt.

7.2.1 Datenaustausch-Mechanismus

Die **Monophase** erfordert keine genauere Erläuterung, da sich diese ausschließlich auf die Arbeitsweise der Z 8001-CPU bezieht. Dies war bereits Gegenstand von Kapitel 2. Die **Initialisierungsphase** ist bezüglich eines Parallelrechners von großer Bedeutung. Während ihrer Aktivität erfolgt ein reger Transfer vom Hauptsystem in die Subsysteme. **Bild 7.2** zeigt die elementaren Vorgänge bei einer Initialisierung. Die beteiligten Baugruppen sind der Kern des Host-Computers (Z 8001 CPU, Z 8010 MMU, Z 8016 DMATC), ferner das main-frame-RAM und bis zu 128 Subsysteme mit Z 8002-CPUs. Die Problematik liegt in vielerlei Punkten.

Es beginnt damit, daß das main-frame-RAM als Quelle physikalisch adressiert wird (über die MMU). Da nun die einzelnen Programme (Teil-Tasks) Umfänge bis 8 KB sehr schnell erreichen können, ist es hier nicht nur sinnvoll, sondern geradezu notwendig, den Transfer einem DMA-Steuerbaustein zu übertragen. Das Problem liegt darin, daß der DMA-Baustein ebenso segmentiert adressieren können muß, wie es die CPU kann. Deshalb ist als DMA-Steuerbaustein nur der Z 8016 DMATC einsetzbar, der über die MMU das RAM physikalisch adressiert. Was bei dieser Art Transfer nur sehr bedingt möglich ist, daß der DMATC sowohl die Quelle (main-frame-RAM) als auch das Ziel (Sub-RAM) einerseits physikalisch und andererseits logisch adressieren kann. Das braucht der DMATC auch gar nicht zu können, denn die Zieladressierung übernimmt ein Adreßgenerator, der in jedem Subsystem integriert ist und zum DMATC synchron geschaltet wird. Die Synchronisierung erfolgt einfach durch die Adreßleitung A0 der logischen DMA-Adresse. Auf den Adreßgenerator wird in Abschnitt 7.3 noch eingegangen.

Der Ablauf der Initialisierungsphase erfolgt nochmals in vier untergliederten Schritten. Die CPU übergibt dem DMATC die Start- und Endadresse eines Teil-Task-Programms, das sich im main-frame-RAM befindet. Der zweite Schritt besteht in der Anregung aller Subsysteme, eine Anforderung auf die Host-CPU auszusenden. Dieser Schritt ist nun beträchtlich komplizierter, als es einfach grafisch darstellbar wäre. Hier muß auf den Abschnitt 7.3 verwiesen werden, weil dort die nötige Hardware vorgestellt wird. Die priorisierte Kette der Subsysteme erzeugt keinesfalls von selbst eine Anregung der Subsysteme, die ihrerseits mit einer Anforderung auf die Host-CPU antworten. Das Ziel dabei ist, ein Subsystem in den Multi-Mikro-Request-Zyklus zu bringen, d.h. $\overline{\mu I}$ abzufragen und bei Erlaubnis $\overline{\mu O}$ zu setzen. Dies erfolgt im Rahmen einer $\overline{NVI}$-Serviceroutine in jedem Subsystem, das durch eine allgemeine Subsystem-Interruptbaugrup-

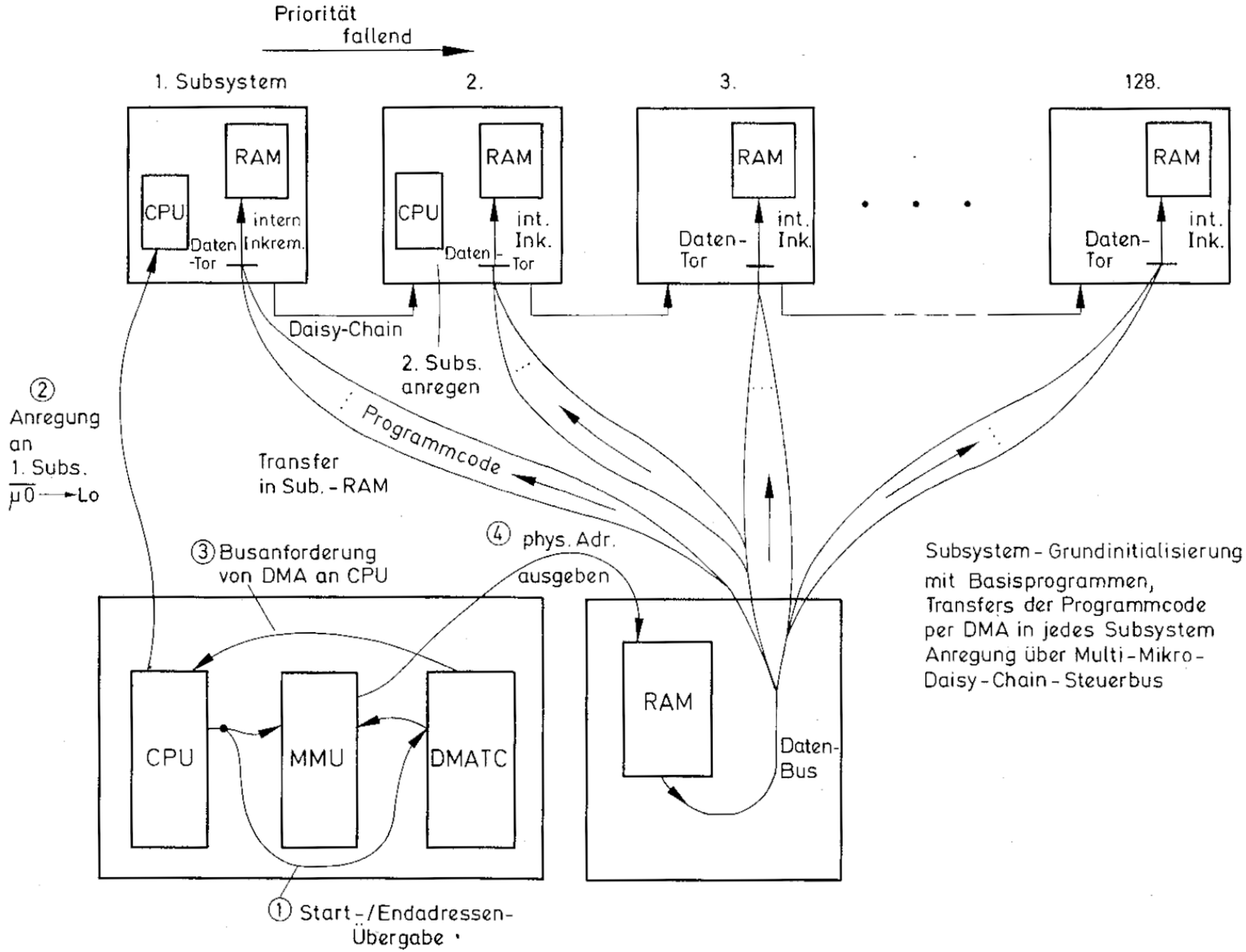

Priorität
fallend
1. Subsystem
2.
3.
128.
RAM
CPU
intern
Inkrem.
Daten
-Tor
int.
Ink.
Daisy-Chain
2. Subs.
anregen
Programmcode
Transfer
in Sub. - RAM
② Anregung an 1. Subs. µ0 → Lo
③ Busanforderung von DMA an CPU
④ phys. Adr. ausgeben
CPU
MMU
DMATC
RAM
Daten-
Bus
① Start-/Endadressen-Übergabe
Subsystem - Grundinitialisierung
mit Basisprogrammen,
Transfers der Programmcode
per DMA in jedes Subsystem
Anregung über Multi-Mikro-
Daisy-Chain-Steuerbus

Bild 7.2

pe (Ausgabe eines $\overline{NVI}$ an jedes Subsystem) ausgelöst wird. Es ist absolut entscheidend, daß sowohl die Reihenfolge als auch die Aktivierung eingehalten wird, weil nach einer Anerkennung eines aktivierten $\overline{\mu O}$ eines Subsystems (und nur eines in der gesamten Kette) das zugehörige Datentor des Subsystems geöffnet wird. Wird das main-frame-RAM an einer Adresse ausgelesen, wird dieses Byte vom DMATC auf eine fingierte Adresse transferiert. Die fingierte Adresse ist natürlich nur scheinbar fingiert, weil der DMATC den Sub-RAM nicht direkt adressieren soll. Die Adressierung übernimmt, wie schon gesagt, der interne Adreßgenerator im Subsystem. Indem der DMATC den Inhalt der Adresse auf den Datenbus bringt, wird dieser über das Datentor dem Sub-RAM zugeführt und an der vom Adreßgenerator ausgegebenen Adresse abgelegt. Die Inkrementierung des Adreßgenerators wird ausschließlich aus A0 des DMATCs abgeleitet. Somit ist der Transfer aus dem main-frame-RAM ab einer individuell vorgebbaren Startadresse möglich. Die Startadresse wird nach einer Link-Operation von der Sub-CPU in den Adreßgenerator initialisiert. Auf diese Weise wird ein Teil-Task in ein Subsystem geladen.

Hat der Adreßgenerator durch Einerschritt-Inkrementierung die Endadresse des Programms erreicht, sendet er an die Sub-CPU einen $\overline{NMI}$. Bei diesem Interrupt muß zwischen zwei Fällen unterschieden werden. In einer internen Subsystem-Initialisierung, die die Sub-CPU autonom aus der eigenen residenten Firmware ausführt, ist eine grundsätzliche Endadresse für Teil-Tasks im Sub-RAM festgelegt (abhängig von der Größe des RAM-Bereichs), die in den Adreßgenerator fest eingeschrieben wird. Wird die Endadresse überschritten, bewirkt der $\overline{NMI}$ das hardwaremäßige Schließen des Datentors und in einer nachfolgenden Serviceroutine die Deaktivierung von $\overline{\mu O}$. Dieser Fall ist grundsätzlicher Art, denn ein RAM ist nur endlich groß und läßt sich nicht überladen. Dieser Vorgang ist, einfach ausgedrückt, ein $\overline{NMI}$ wert. Der $\overline{NMI}$ für das Sub-RAM kann jedoch auch von einer externen Logik des Host-Computers kommen, d.h. genaugenommen wird das die Regel sein. Wird eine Programmendadresse irgendwo unterhalb der kritischen Schwelle erreicht, löst dieser $\overline{NMI}$ die gleiche Sub-Serviceroutine aus, in der das Datentor zuerst hardwaremäßig "dicht" gemacht und dann per Programm eine Zugriffserlaubnis an das nächst niedrigere Subsystem weitergegeben wird. Die Beendigung der $\overline{NMI}$-Serviceroutine kann zugleich auch ein Start der Teil-Task-Ausführung sein. Anzumerken ist, daß unmittelbar nach der Anerkennung des $\overline{NMI}$ die im hochohmigen Zustand befindliche CPU (aufgrund des $\overline{NVI}$) sich selbst hardwaremäßig aktiviert. Das erfolgt dadurch, indem sich der Status Nr. 6 in Nr. 5 ändert und damit den Bus-Request der CPU löst. Im weiteren wird hier auf Abschnitt 7.3 verwiesen, der die Hardware dafür zeigt.

Die Vorgänge im Subsystem sind an dieser Stelle elementar beschrieben, weil sie im gesamten Ablauf doch ziemlich kompliziert sind. Die Initialisierungsphase beruht auf einem diffizilen Interrupt-Handling, gekoppelt mit einem Bus-Request. Die Datentor-Freigabe bzw. -Öffnung ist einem Multi-MIkro-Daisy-Chain-Anerkennungszyklus eingeordnet. Die Reihenfolge ist folgendermaßen:

1. : Ausgabe eines $\overline{NVI}$ aus einer übergeordneten Interruptlogik an alle Subsysteme.
   Wirkung: Jedes Subsystem regt daraufhin einen Anforderungszyklus an, der prinzipiell in Abschnitt 2.11 behandelt wurde.
2. : Ist eine Anforderung anerkannt worden, wird die Sub-CPU nach einigen vorbereitenden Operationen durch $\overline{BUSRQ}$ gleich Lo abgeschaltet. Erst nach dieser Abschaltung wird das Datentor geöffnet, d.h. der Sub-RAM kann jetzt geladen werden.

Nach der Ladung des Sub-RAMs wird der interne Sub-Bus der CPU wieder

übergeben, wie kurz zuvor angedeutet wurde.
Die **Parallelphase** ist unkritisch, weil hier die Subsysteme unabhängig voneinander arbeiten.
Die **Datenhol- und Austauschphase** erfolgt ziemlich ähnlich wie die Initialisierungsphase. Der Sub-RAM wird jedoch gelesen und einzelne Adreßinhalte in den main-frame transferiert. Diese Transfer-Operation wird direkt von der Z 8001-Host-CPU ausgeführt. Dies ist deshalb so gedacht, weil die Quantität der zu holenden Bytes (Ergebnisse) viel geringer als bei der Initialisierung ist. Mit den leistungsfähigen Stringoperationen der CPU ist dies sehr leicht machbar.

Wenn in der bisherigen Darstellung des Datenaustausch-Mechanismus die Vorgänge relativ einfach angegeben wurden, ist dies ausschließlich der raffinierten Hardware zu verdanken, die durch die Z 8000-Bausteine und den vielfach verwendeten AMD-Logikbausteinen gegeben ist.

Es muß an dieser Stelle nochmals darauf hingewiesen werden, daß es sich hier um die Beschreibung eines absoluten Größtrechners handelt. Für bestimmte Aufgaben, die sich sehr gut parallelisieren lassen und die Effektivität der Parallelisierung ebenso hoch ist, läßt sich mit einem Z 8000-Parallelrechner ein Datendurchsatz erreichen, neben dem eine CYBER 176 von Control Data nicht mehr mithalten kann.
Im folgenden wird auf die Hardware dieses Z 8000-Monstrums noch näher eingegangen.

7.3 HARDWARE EINES Z 8000-PARALLELRECHNERS NACH DEM "MASTER-SLAVE"-PRINZIP

Das konzipierte Parallelrechner-System ist für maximal 128 Subsysteme ausgelegt. Diese Zahl ist sehr einfach begründbar. Jedes 64 K-Segment der Host-CPU wird von einem Subsystem belegt. Die adreßfeldmäßige Trennung ist ebenso einfach. Das 8 MB-Feld der logischen Adreßfeldaufteilung für alle Subsysteme wird grundsätzlich mit dem Status Nr. 8 (Anforderungszyklus für Datenspeicher) angewählt. Das main-frame-Speicherwerk (RAM und ROM) wird praktisch in allen drei Arten angewählt: Anforderung auf Programmcode (IF1, IFn), Anforderun- auf Datenspeicher und Stapel. Wie sind nun alle Baugruppen des Parallelrechners verbunden? Im folgenden Unterabschnitt wird dies gezeigt.

7.3.1 Multibus-System

Der Begriff Multibus steht für Multiprozessor-Systembus und beinhaltet den 16-bit-Adreßbus, den 16-bit-Datenbus, den Steuerbus (bestehend aus $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, B/$\overline{W}$, $\overline{MREQ}$, N/$\overline{S}$, AØ und eventuellen Statusleitungen) und letztlich dem Multi-Mikro-Daisy-Chain-Bus. Das gesamte Parallelrechnersystem ist in seiner Struktur in **Bild 7.3** dargestellt. Wie zu sehen ist, besteht es aus vier Host-Computerkomponenten (untere Reihe) und den 128 Subsystemen (obere Reihe). Der Host-Computer besteht aus CPU + Hauptspeichersystem (main-frame) + Interrupterzeugung + Multifunktions-Dialog-E/A-Schnittstelle. Die gesamten Verbindungen stellt der besagte Multibus her. Der physikalische Adreßbus der MMU (A8 . . . A23) zum Hauptspeicher und die Segmentadreßleitungen SN0 . . . SN6 von der Host-CPU zur Interrupterzeugung gehört, streng genommen, noch dazu, wenn die Verbingungen auch nur mittelbar mit den Subsystemen zu tun haben. Im folgenden Abschnitt werden die einzelnen Systembaugruppen genauer vorgestellt.

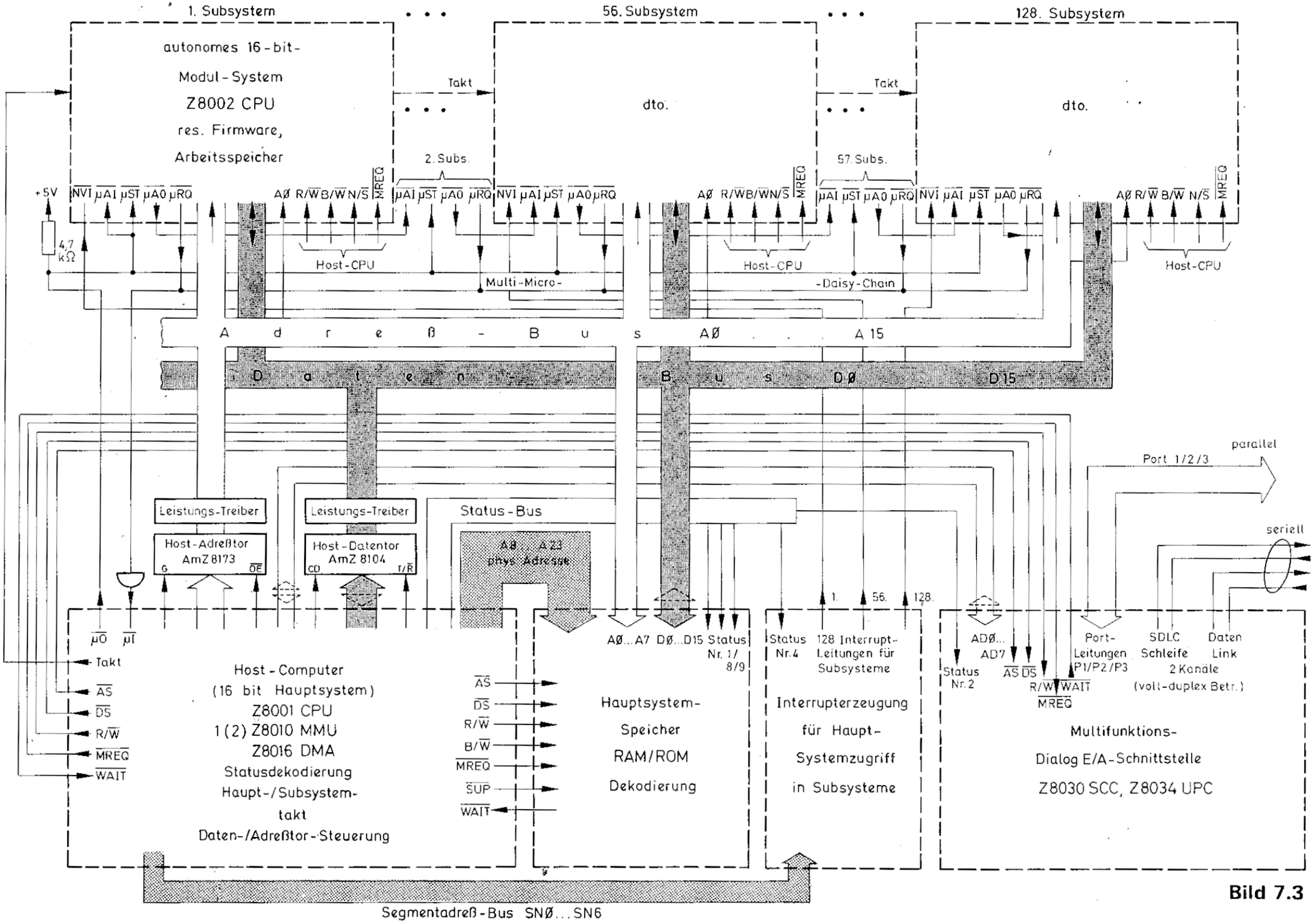

1. Subsystem
56. Subsystem
128. Subsystem
autonomes 16-bit-
Modul-System
Z8002 CPU
res. Firmware,
Arbeitsspeicher
dto.
dto.
Takt
Takt
2. Subs.
57. Subs.
+5V
4,7 kΩ
NVI µAI µST µA0 µRQ
AØ R/W B/W N/S MREQ
Host-CPU
Multi-Micro-
-Daisy-Chain
Adreß-Bus AØ . . . A 15
Daten-Bus DØ D15
Leistungs-Treiber
Leistungs-Treiber
Status-Bus
Host-Adreßtor AmZ 8173
Host-Datentor AmZ 8104
AØ...A23 phys Adresse
parallel
Port 1/2/3
seriell
Takt
AS
DS
R/W
MREQ
WAIT
Host-Computer
(16 bit Hauptsystem)
Z8001 CPU
1(2) Z8010 MMU
Z8016 DMA
Statusdekodierung
Haupt-/Subsystem-
takt
Daten-/Adreßtor-Steuerung
Hauptsystem-
Speicher
RAM/ROM
Dekodierung
AØ...A7 DØ...D15 Status Nr. 1/ 8/9
Status Nr.4
128 Interrupt-Leitungen für Subsysteme
Interrupterzeugung
für Haupt-
Systemzugriff
in Subsysteme
ADØ... AD7
Status Nr.2
Port-Leitungen P1/P2/P3
SDLC Schleife 2 Kanäle (voll-duplex Betr.)
Daten Link
Multifunktions-
Dialog E/A-Schnittstelle
Z8030 SCC, Z8034 UPC
Segmentadreß-Bus SNØ...SN6

Bild 7.3

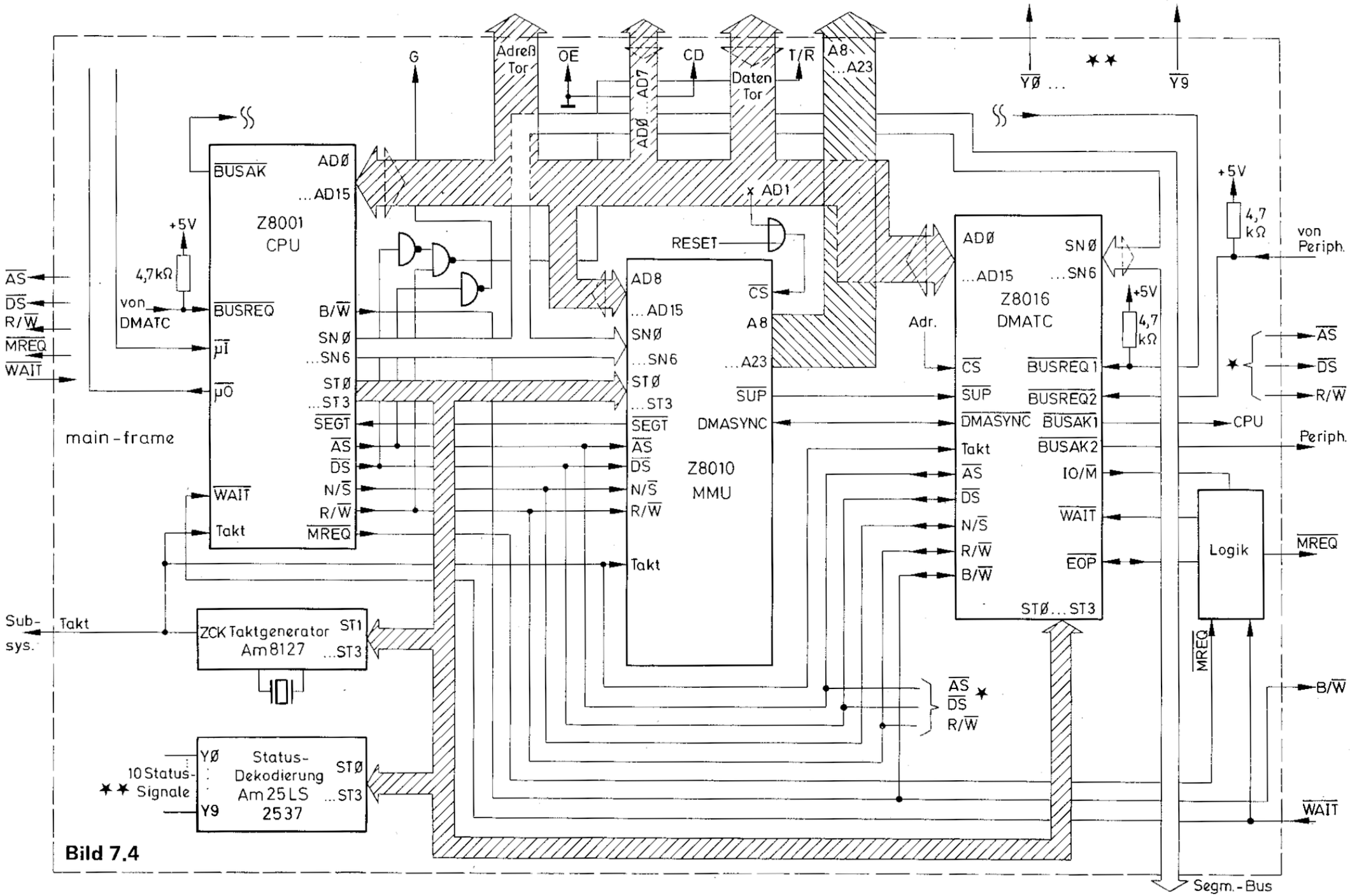

Adreß Tor
OE
G
AD7 ... AD0
CD
Daten Tor
T/R
A8 ... A23
YØ ...
Y9
Z8001 CPU
BUSAK
ADØ
...AD15
+5V
4,7kΩ
von DMATC
BUSREQ
B/W
SNØ
...SN6
STØ
...ST3
SEGT
AS
DS
N/S
R/W
MREQ
WAIT
Takt
µI
µO
AS
DS
R/W
MREQ
WAIT
main-frame
RESET
x AD1
AD8
...AD15
SNØ
...SN6
STØ
...ST3
SEGT
AS
DS
N/S
R/W
Takt
CS
A8
...A23
SUP
DMASYNC
Z8010 MMU
Adr.
ADØ
...AD15
SNØ
...SN6
Z8016 DMATC
CS
SUP
DMASYNC
Takt
AS
DS
N/S
R/W
B/W
BUSREQ1
BUSREQ2
BUSAK1
BUSAK2
IO/M
WAIT
EOP
STØ...ST3
+5V
4,7kΩ
von Periph.
AS
DS
R/W
CPU
Periph.
Logik
MREQ
B/W
WAIT
Segm.-Bus
Sub-sys.
Takt
ZCK Taktgenerator Am8127
ST1
...ST3
AS
DS
R/W
10 Status-Signale
YØ
Y9
Status-Dekodierung Am25LS 2537
STØ
...ST3

Bild 7.4

7.3.2 Komponenten des Rechners

Die Baugruppe des Host-Computer-Kerns zeigt **Bild 7.4.** Diese Schaltung zeigt eine Ähnlichkeit zu der in Bild 5.27. Für den Parallelrechner wurde eine MMU eingesetzt, weil damit der Bereich des main-frame-Speicherwerks ausreichend organisiert werden kann. Der Parallelrechner ist nicht als Mehrplatzsystem gedacht und benötigt aus diesem Grund keinen so extensiven Speicherraum. Deshalb ist die Organisation von 64 Segmenten ausreichend. Die oberste Maxime, die diesem Rechner zu Grunde liegt, ist "Speed, Speed und nochmals Speed". Dafür muß jedes Mittel recht sein. Da man es bei einem Rechner dieser Größenordnung mit teilweise sehr umfangreichen Programmen zu tun bekommt, kann andererseits auf die Unterstützung der MMU nicht verzichtet werden. Eine MMU ist daher ein sehr wohl annehmbarer Kompromiß.
Es gibt noch einen weiteren Grund. Man kann beispielsweise die MMU auf das Feld der Segmente 64 . . . 128 definieren (URS-Flag gleich 1). Somit bleiben die unteren 64 Segmente frei. Benützt man nun diesen logischen Adreßbereich bis 4 Megabyte für die Subsysteme, ist die Feldauswahl sehr einfach, weil unterhalb von 4 MB die MMU inaktiv bleibt, d.h. der main-frame-RAM wird nicht angesprochen. Wünscht man mit der Host-CPU unmittelbar in irgendeinen Sub-RAM zu schreiben, kommt man überhaupt nicht mit dem physikalischen Adreßbereich in Konflikt. Auf diese Weise läßt sich für die ersten 64 Subsysteme eine problemlose Adreßfeldgliederung innerhalb von 0 . . . 4 MB definieren. Arbeitet man im Monobetrieb mit der Host-CPU allein, steht das logische Feld zwischen 4 . . . 8 MB zur Verfügung, das durch die MMU auf 16 MB gespreizt wird.

Sobald mit mehr als 64 Subsystemen gleichzeitig gearbeitet wird, muß der Zugriffs-Konfliktfall betrachtet werden.

Wie sieht der aus?
Betreffend ist der Konfliktfall ohnehin nur in der Initialisierungsphase bzw. Austauschphase. Der DMATC adressiert in Lesezyklen den main-frame-RAM und in fingierten Schreibzyklen einen Sub-RAM. An das betreffende Sub-RAM gelangt aber nur R/$\overline{W}$ gleich Lo und, ausgenommen A0, keine weitere Adreßleitung, weil das, wie schon angedeutet, der interne Subsystem-Adreßgenerator erledigt. Er adressiert primär den Sub-RAM, so daß im eigentlichen Transferfall der main-frame-RAM ausgelesen und zugleich der Sub-RAM beschrieben wird, – mit zwei Adressen aus zwei unterschiedlichen Hardwarequellen. Es gibt auch hier keine Probleme bezüglich der Kompetenz einer Adresse. Das eigentlich neue an dem Parallelrechner ist der Adreßgenerator. Er wird von dem Z-8016 DMATC synchronisiert, indem er ihn über A0 inkrementiert. Dies ist bei Leibe nichts elektronisch futuristisches.

Der einzig wirkliche Konfliktfall kann in der Austauschphase auftreten, wenn die Host-CPU im 4 . . . 8-MB-Feld der Sub-RAM-Bereiche liest und den Inhalt in das main-frame-RAM übertragen will, d.h. aus Subsystemen der Nummern zwischen 64 . . . 128. Hier ist eine zusätzliche Hardware notwendig, die das gleichzeitige Lesen einer physikalischen Speicherstelle verhindert, während eine Sub-RAM-Adresse gelesen wird. Es muß ein Inhibit für eine physikalische Adreßausgabe oder eine Freigabeunterdrückung erzeugt werden. Dies kann an Adreßtreibern oder am einfachsten direkt an den Adreßdekodierern erfolgen. Der Verlauf des Transfers vom Sub-RAM in ein Ziel im main-frame-RAM erfolgt über die CPU-Register. Sicher ist das etwas umständlich; sind z.B. nur 50 Bytes zu holen, lohnt sich jedoch ein DMA-Einsatz nicht.
Als Anregung sei an dieser Stelle eine hardwaremäßige Lösung genannt, die die Austauschphase sehr vereinfachen kann. Man setzt einen oder mehrere E/A-FiFo-Bausteine in jedes Subsystem mit ein. Die Sub-CPU transferiert nach jedem ausgeführten Sub-Task die jeweili-

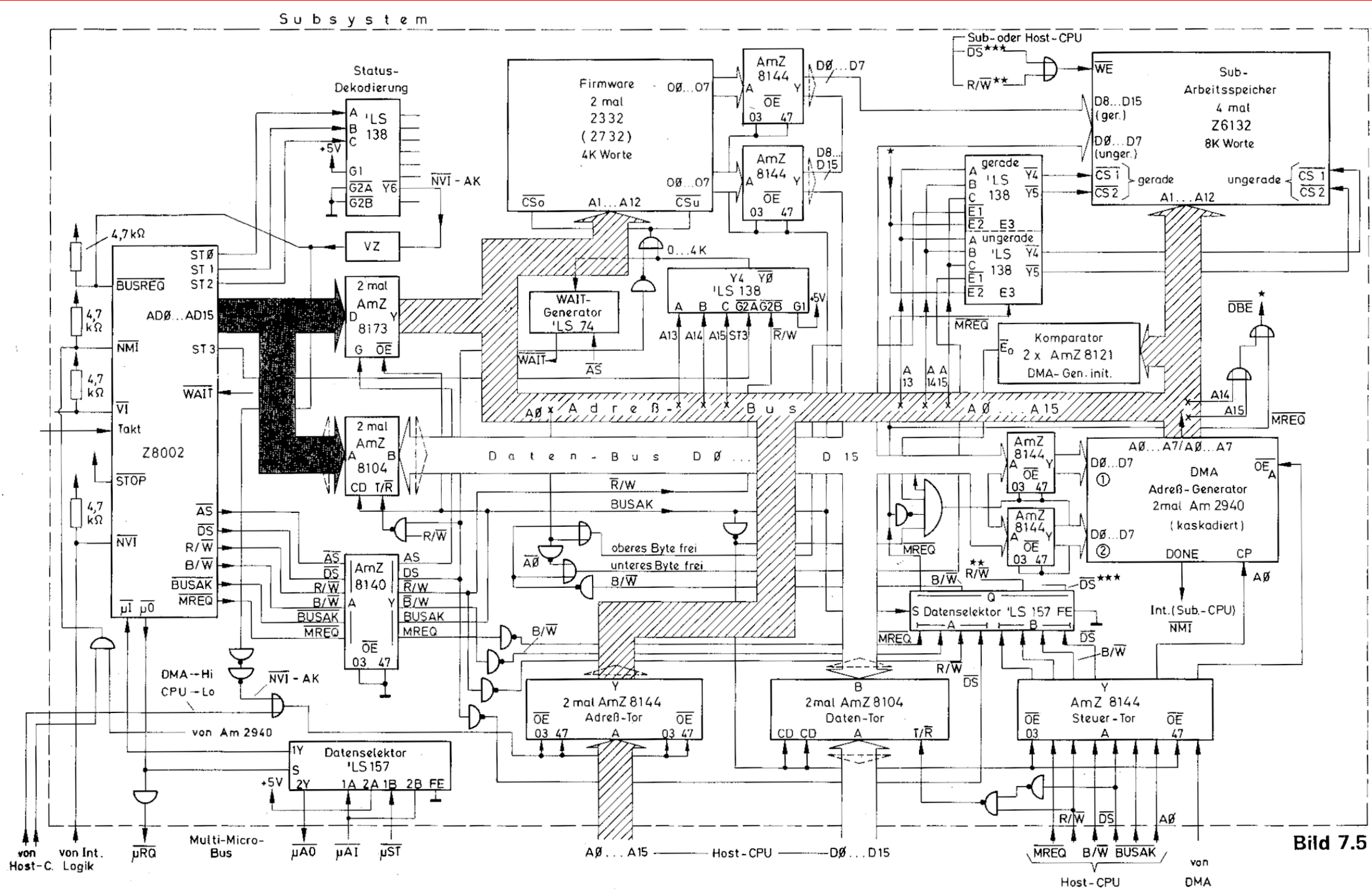

Subsystem
Sub- oder Host-CPU
DS***
R/W**
Status-
Dekodierung
Firmware
2 mal
2332
(2732)
4K Worte
AmZ
8144
Sub-
Arbeitsspeicher
4 mal
Z6132
8K Worte
WE
D8...D15
(ger.)
DØ...D7
(unger.)
gerade
ungerade
A1...A12
'LS
138
VZ
NVI - AK
4,7 kΩ
STØ
ST 1
ST2
BUSREQ
ADØ...AD15
NMI
ST3
WAIT
VI
Takt
Z8002
STOP
AS
DS
R/W
B/W
BUSAK
MREQ
μI μ0
2 mal
AmZ
8173
2 mal
AmZ
8104
CD T/R
AmZ
8140
WAIT-
Generator
'LS 74
WAIT
AS
0...4K
Y4 YØ
'LS 138
A13 A14 A15 ST3 R/W
Komparator
2 x AmZ 8121
DMA-Gen. init.
DBE
A14
A15
MREQ
AØ x A d r e ß - x x x B u s AØ ... A15
D a t e n - B u s DØ ... D 15
R/W
BUSAK
oberes Byte frei
unteres Byte frei
B/W
AØ
DMA
Adreß-Generator
2mal Am 2940
(kaskadiert)
DØ...D7
DONE
CP
Int.(Sub.-CPU)
NMI
Datenselektor 'LS 157 FE
2 mal AmZ 8144
Adreß-Tor
2mal AmZ 8104
Daten-Tor
AmZ 8144
Steuer-Tor
DMA-Hi
CPU-Lo
von Am 2940
Datenselektor
'LS157
+5V
μRQ
μA0
μA1
μST
Multi-Micro-
Bus
von
Host-C.
von Int.
Logik
AØ...A15 —— Host-CPU —— DØ...D15
Host-CPU
von
DMA

Bild 7.5

gen Ergebnisse in dieses FiFo-RAM bzw. Register. Die Host-CPU liest den Inhalt mit Hilfe von E/A-Stringbefehlen direkt in den main-frame-RAM. Die passenden Bausteine sind der Z 8038 FIO und der Z 8060 FIO-Buffer, die zusammen individuell kaskadiert werden können. Der Vorteil der Methode ist eine Holphase aus dem Subsystem, die vollkommen asynchron und damit zeitunabhängig ablaufen kann. Alle kurzen Austauschvorgänge wären über die FiFos abzuwickeln, wogegen die langwierigen Initialisierungsvorgänge, wie gehabt, per DMA geschehen. Das brächte einen weiteren Zeitgewinn. Als Nachteil ist nur zu nennen, daß sich die Hardware nochmals vergrößert und verteuert.

Bild 7.5 zeigt die komplette Schaltung eines Subsystems mit der Z 8002-CPU. Das Netzwerk ist im ersten Moment etwas verwirrend, zumal das Bild beträchtlich verkleinert werden mußte. Das System besteht jedoch aus wesentlichen Grundschaltungen, wie sie in Kapitel 4 behandelt wurden, d.h. es sind in der Tat nur wenig neue Elemente eingebaut. Diese sind im einzelnen untereinander genannt:

- 16-bit-Adreßgenerator mit zwei Bausteinen des Typs Am 2940
- 4-bit-Datenselektor LS 157 zur Umschaltung von Sub-CPU-Steuersignale $\overline{MREQ}$, B/$\overline{W}$, R/$\overline{W}$ und $\overline{DS}$ auf die identischen der Host-CPU
- 4-bit-Datenselektor LS 157 zum Aufbau des Multi-Mikro-Daisy-Chain-Busses
- Jeweils zweimal die Bausteine AmZ 8144 bzw. AmZ 8104 für das Adreß- und Datentor zum Multibus bzw. zur Host-CPU
- Die Statusdekodierung des $\overline{NVI}$ mit einer nachfolgenden Verzögerungsleitung (Laufzeit $\approx$ 10 μs) an den Bus-Request-Eingang der CPU.

Diese hardwaretechnischen Elemente sind unbedingt für einen "Slave"-Betrieb notwendig. Was das Speicherwerk betrifft, sind die Angaben als relativ, aber ebenso realistisch zu betrachten. Die Grenzen sind hardwaremäßig vorgegeben (64 KB).

Adreßgenerator: Dieses Sub-Element wird von der Sub-CPU direkt initialisiert. Dieser Baustein ist ein relativ kompliziertes Zählerelement. **Bild 7.6** zeigt die Schaltung des Adreßgenerators als genaues Detail in der Subsystem-Hardware. Zwei Bausteine ergeben eine Adreßbreite von 16 bit. Der Am 2940 ist ein 28-poliger, sehr schneller Bipolar-Baustein, der über einen 8-bit-Adreßzähler verfügt. Die Speicheradresse ist über $\overline{ACI}$ inkrementierbar oder dekrementierbar und liefert über $\overline{ACO}$ einen Übertrag an den nächsten Baustein. In das Adreßregister wird die Startadresse geladen und im Wortzähler wird die Anzahl der zu übertragenden Worte gespeichert. Ist eine Koinzidenz zum aktuellen Adreßzählerstand und der Endadresse erreicht worden, wird der Ausgang DONE aktiviert, der direkt an den $\overline{NMI}$ des Z 8002 geleitet wird. Die daraufhin eingeleitete Serviceroutine schließt das Adreß- und Datentor.

Der Beginn des DMA-Transfers von main-frame-RAM in das Sub-RAM erfolgt sobald, nachdem die $\overline{NVI}$-Ausgabe der Host-Interruptlogik eine Anerkennung der Busanforderung eines Subsystems auf den Multibus nach entsprechender Verzögerung ausgelöst hat. Hat ein Subsystem das Vorrecht auf den Multibus erlangt, müssen sofort (innerhalb des Taktzyklus) die $\overline{OE_A}$-Eingänge beider 2940 freigegeben werden. Dies muß extern synchron zum Start des Transfers erfolgen. Daraufhin sind die Adreßausgänge beider 2940-Bausteine frei und adressieren das Sub-RAM. Der Adreßbus von der CPU war bereits hochohmig geschaltet.

Der Baustein Am 2940 ist wie geschaffen für diesen Zweck. Es gibt zum Original von AMD keinen äquivalenten Typ einer anderen Firma, der die gleichen Fähigkeiten aufweist.

Statusdekodierung des $\overline{NVI}$: Dieses winzige Detail des Subsystems betrifft eine

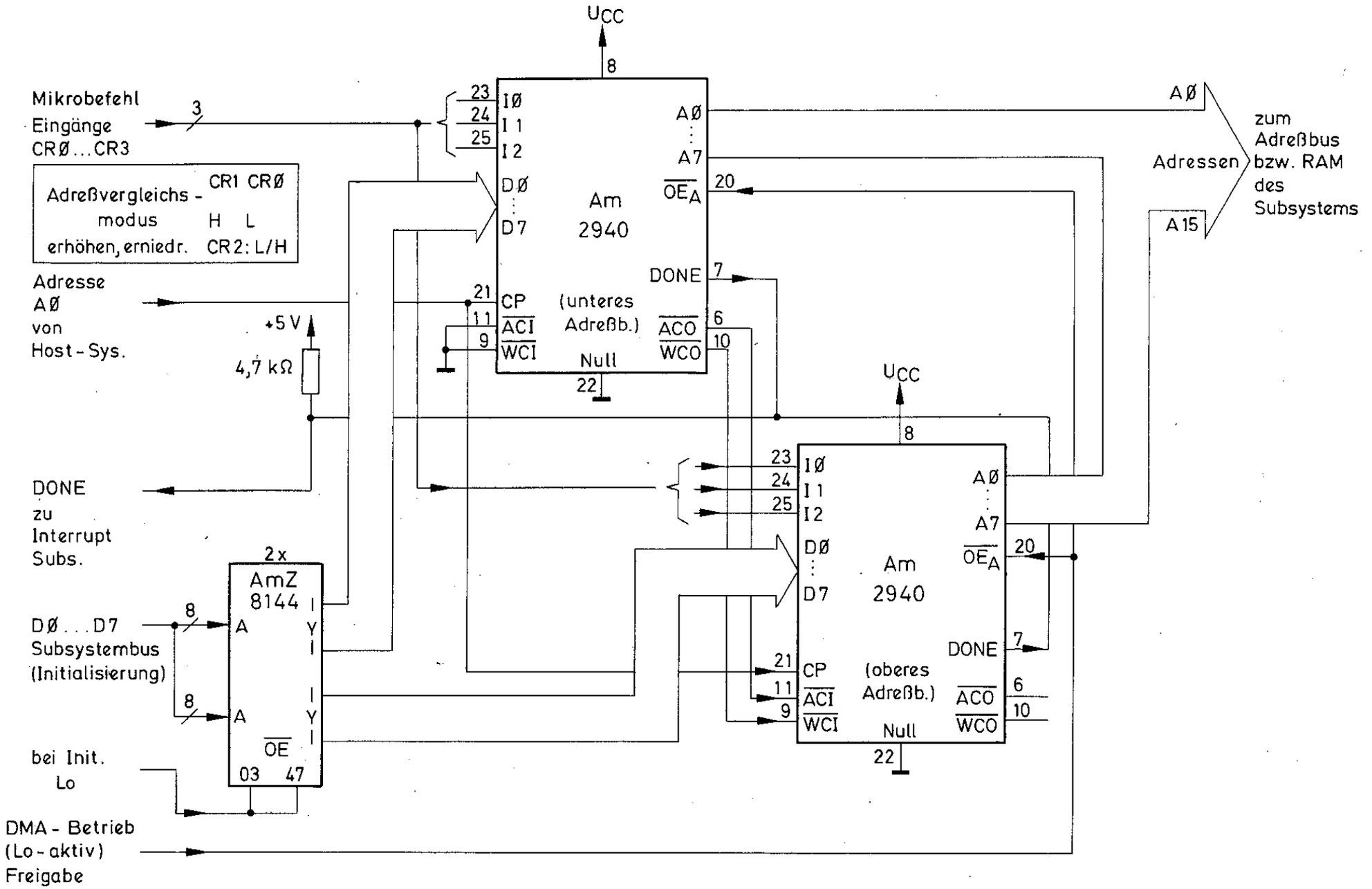

UCC
8
Mikrobefehl
Eingänge
CRØ...CR3
3
Adreßvergleichs - modus
erhöhen, erniedr.
CR1 CRØ
H L
CR2: L/H
Adresse
AØ
von
Host - Sys.
+5 V
4,7 kΩ
DONE
zu
Interrupt
Subs.
2x
AmZ
8144
DØ...D7
Subsystembus
(Initialisierung)
8
bei Init.
Lo
DMA - Betrieb
(Lo - aktiv)
Freigabe
Am
2940
(unteres
Adreßb.)
IØ
I1
I2
DØ
D7
AØ
A7
OEA
CP
ACI
WCI
ACO
WCO
DONE
Null
23
24
25
21
11
9
20
7
6
10
22
(oberes
Adreßb.)
AØ
Adressen
A15
zum
Adreßbus
bzw. RAM
des
Subsystems

Bild 7.6

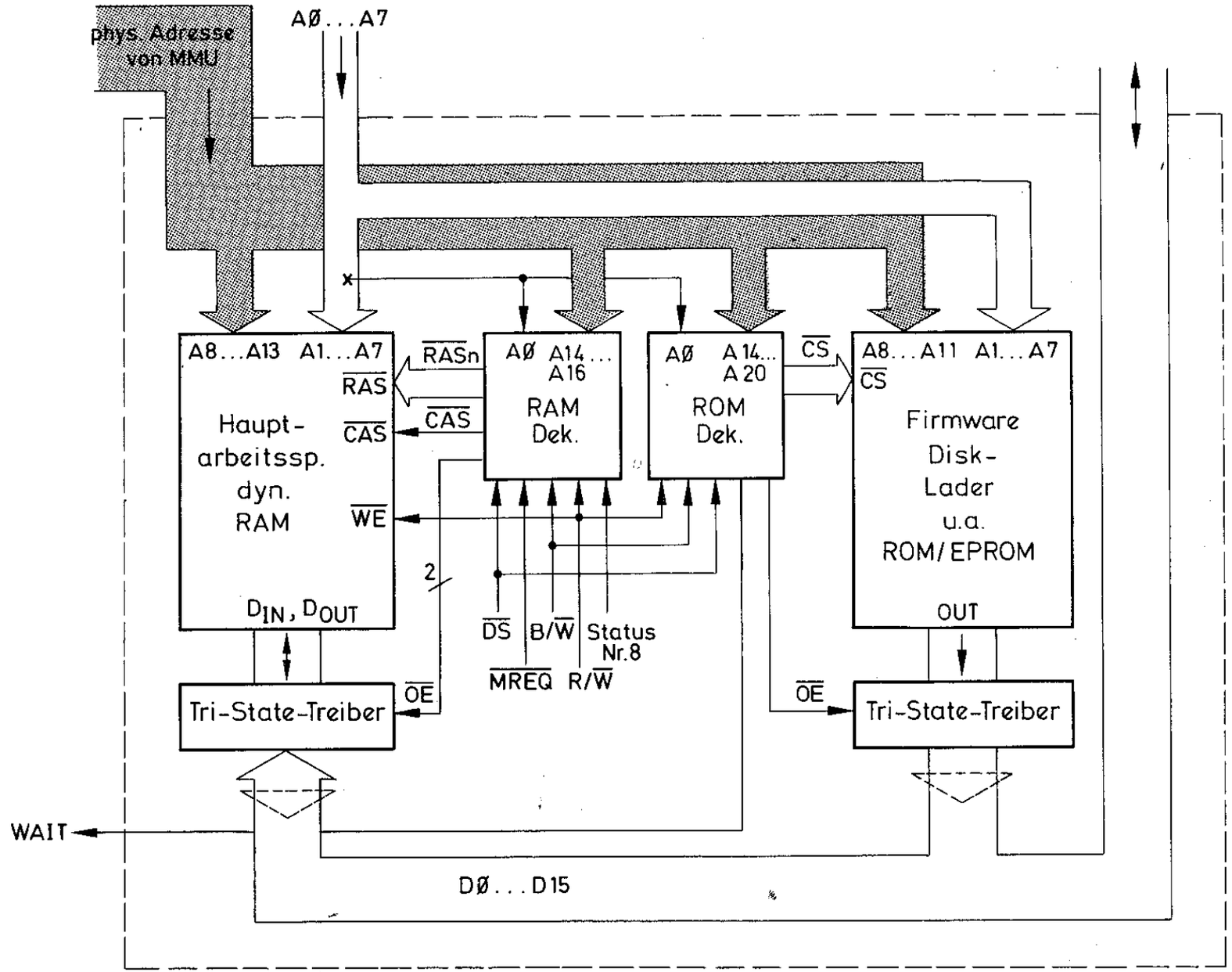

phys. Adresse von MMU
AØ...A7
A8...A13 A1...A7
RAS
CAS
WE
Haupt-arbeitssp. dyn. RAM
DIN, DOUT
RASn
AØ A14... A16
RAM Dek.
AØ A14... A20
ROM Dek.
CS
A8...A11 A1...A7
CS
Firmware Disk-Lader u.a. ROM/EPROM
OUT
2
DS B/W Status Nr.8
MREQ R/W
OE
Tri-State-Treiber
OE
Tri-State-Treiber
WAIT
DØ...D15

Bild 7.7

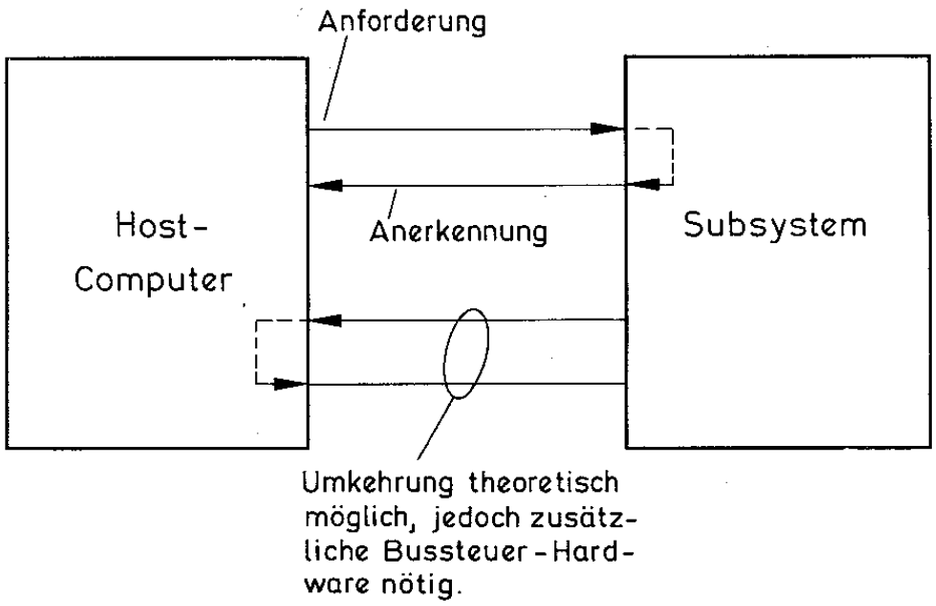


Bild 7.8

in der Tat komplizierte Funktion. Nach einem $\overline{NVI}$ aus der übergeordneten Host-Interruptlogik erfolgt eine übliche Anerkennung durch die Sub-CPU, das sich in einem Lo an $\overline{Y6}$ der Statusdekodierung äußert (bis dahin nicht kompliziert). Der $\overline{NVI}$-Status wird dazu benützt, um einerseits die CPU abzuschalten (an $\overline{BUSRQ}$ eine Übergabe anfordern) und andererseits das Adreßtor nach einer Bedingung zu öffnen. Das ODER-Gatter unterhalb der CPU benötigt eine sehr diskrete Steuerung, die vom Hostsystem präzise getimet erfolgen muß. Darin liegt dieses Hardwareproblem. Bei einer Initialisierungsphase darf das Adreßtor nicht geöffnet werden, weil bei einem DMA-Transfer in den Sub-RAM der Adreßausgang der beiden 2940 aktiv ist, d.h. am ODER-Gatter muß der zweite Eingang Hi sein, womit eine Initialisierungsphase damit angezeigt wird. Wird hingegen ein direkter Zugriff der Host-CPU in den Sub-RAM eingeleitet, muß am zweiten ODER-Gattereingang ein Lo anliegen, wodurch der Ausgang auf Lo liegt und damit das Adreßtor geöffnet ist. Das ist ganz einfach deshalb so, weil bei einer Austauschphase mit der Host-CPU diese zugleich selbst den Sub-RAM adressiert. Der Adreßgenerator bleibt inaktiv. Das Signal für den besagten ODER-Gattereingang ist demzufolge sehr präzise pegelrichtig vom Host-Computer zu liefern. Damit ist die komplizierte Funktion gemeint.

Hauptspeicherwerk: In **Bild 7.7** ist das gesamte main-frame-Speicherwerk dargestellt, das das Hauptresource des Parallelrechners betrifft. Es besteht aus einem sehr massiven, dynamischen RAM-Speicherwerk; unter massiv ist die Größe gemeint, die bis zu 4 MB betragen kann. Die Firmware in Festwertspeichern ist abhängig davon, ob das Hostsystem von einem Betriebssystem ausgeht, das auf Plattenspeichern eingelagert ist. In der Regel wird es sich so verhalten. Daraus folgt, daß sich die Firmware in residenten Halbleiterspeichern auf ein Minimum begrenzt. Was in jedem Fall resident in ROMs eingelagert ist, bezieht

sich auf die Grundinitialisierung der Host-CPU, der MMU, der Schnittstellenbausteine und zuletzt einer Firmware für Hard-Disk-Transfers per SDLC oder parallel. Der residente Bereich im mainframe-Speicherwerk wird also verschwindend gering gegenüber dem RAM-Feld sein. Wie aus dem Bild weiter zu ersehen ist, wird der Hauptspeicher ausschließlich physikalisch adressiert. Die Treiberbausteine am Datenbus des Speichers sind auf der anderen Seite am System-Multibus angeschlossen.

Host-Interruptlogik für die Subsysteme: Es wurde bereits mehrfach angedeutet: Der Host-Computer macht alles. Er ist der große Meister, ohne den nichts geht. Fordert der Host-Computer ein Subsystem an, kann das nur er und nicht umgekehrt. Die Konzeption des Parallelrechners legt diesen Sachverhalt so fest (**Bild 7.8**). Die Gründe liegen in der einfacheren Hardware und in der Speed.

Die Einleitung einer Anforderung auf ein Subsystem geschieht mit Hilfe des $\overline{\text{NVI}}$, der synchron von der Interruptlogik erzeugt wird. In der Initialisierungsphase erfolgen maximal 128 synchrone, nicht vektorisierte Interrupts, die die Host-CPU auslöst. In der Datenhol- und Austauschphase ist es ein einziger, der an ein bestimmtes Subsystem geschickt wird. Für diese häufig auftretende Arbeitsphase des Parallelrechners geben die **Bilder 7.9** und **7.10** einen Überblick über die entsprechenden Vorgänge. Sie zeigen jeweils das Schreiben und Lesen vom Hauptspeicher in den Sub-RAM bzw. umgekehrt. Die Hardware der Interrupterzeugung wird vereinfacht in **Bild 7.11** dargestellt. Über die Segmentleitungen SN0 . . . SN6 wird eine Segmentadresse der Host-CPU in einen 4 zu 16 und 3 zu 8 Dekoder aufgesplittet. Die Segmentadresse wird in einem AmZ 8173 zwischengespeichert. Dies hat einen ganz besonderen Grund. Da ein $\overline{\text{NVI}}$ entsprechend lange, u.U. bis zu 2 µs an dem CPU-Eingang angelegt sein muß, bis er anerkannt ist, muß dieser Verzögerung Rechnung getragen werden. Die dazu gehörige Hardwarehilfe ist ein Monoflop. Mit ihm wird die Verzögerung erzeugt, die einen ausgewählten Interrupt etwa 2 µs aktiv sein läßt, d.h. er bleibt so lange am CPU-Eingang aktiv angelegt. In der Schaltung nach **Bild 7.12** wird noch genauer gezeigt, wie die $\overline{\text{NVI}}$-Erzeugung generiert wird.

Sie funktioniert nach dem Matrixprinzip. In dem Baustein LS 154 werden die vier niederwertigen Segmentadreßleitungen SN0 . . . SN3 in 16 diskrete Signale dekodiert. Diese Leitungen werden auf jeweils eine Zweiergruppe nicht invertierender Tri-State-Treiber des Typs AmZ 8144 angelegt. Der zweite Dekodierer liefert acht diskrete Signale aus den Segmentadreßleitungen SN4 . . . SN6. Diese führen von dem LS 138 direkt an die Tri-State-Freigabeeingänge. Eine Leitung gelangt jeweils an eine Zweiergruppe von AmZ 8144. Ist eine der acht Gruppen aktiviert, wird an einem ihrer 16 Ausgänge ein Lo ausgegeben. Dieses Lo an einem Y-Ausgang ist eine diskrete $\overline{\text{NVI}}$-Auslösung für ein Subsystem. Der Ausgang ist, wie schon angedeutet, lange genug aktiv, bis der $\overline{\text{NVI}}$ durch die Sub-CPU anerkannt ist.

Die Initialisierungsphase erfordert die sequentielle Auslösung einer Zahl von Interrupts, die der Zahl notwendiger Subsysteme für eine Prozedur entsprechen. Die Auslösung kann folgendermaßen geschehen:

Mit Hilfe eines E/A-Befehls wird von der Host-CPU ein Segmentadreßzähler geladen. Für diesen Zweck ist ebenfalls wieder der Typ Am 2940 sehr gut verwendbar. Seine Ausgänge werden an den Segmentadreßbus vor den Dekodiereingängen (Bild 7.11) verbunden. Der Wortzähler des 2940 wird mit der Zahl der benötigten Subsysteme (0 . . . 127) geladen. Der Adreßzähler wird auf Null gesetzt. Die Interrupterzeugung wird gestartet, indem der Adreßzähler mit einem Takt von etwa 3 µs inkrementiert wird (ausreichend lange Zeit der Aktivzeit eines $\overline{\text{NVI}}$). Ist nach fortlaufendem Inkrementieren der Wert im Wortzähler

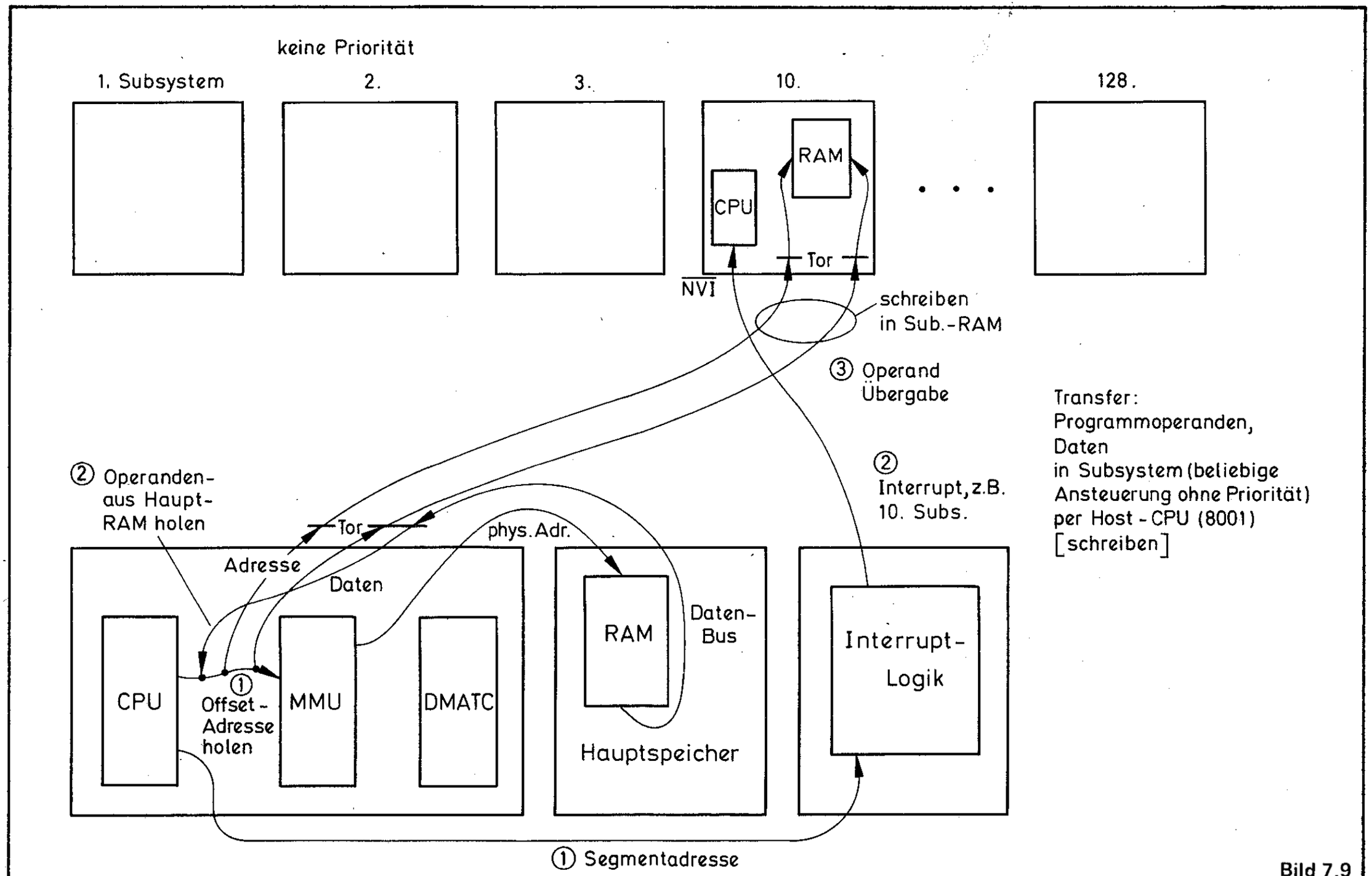

keine Priorität
1. Subsystem
2.
3.
10.
128.
RAM
CPU
Tor
NVI
schreiben
in Sub.-RAM
③ Operand
Übergabe
② Operanden-
aus Haupt-
RAM holen
Tor
phys. Adr.
Adresse
Daten
② Interrupt, z.B.
10. Subs.
Transfer:
Programmoperanden,
Daten
in Subsystem (beliebige
Ansteuerung ohne Priorität)
per Host - CPU (8001)
[schreiben]
CPU
① Offset-
Adresse
holen
MMU
DMATC
RAM
Daten-
Bus
Hauptspeicher
Interrupt-
Logik
① Segmentadresse

Bild 7.9

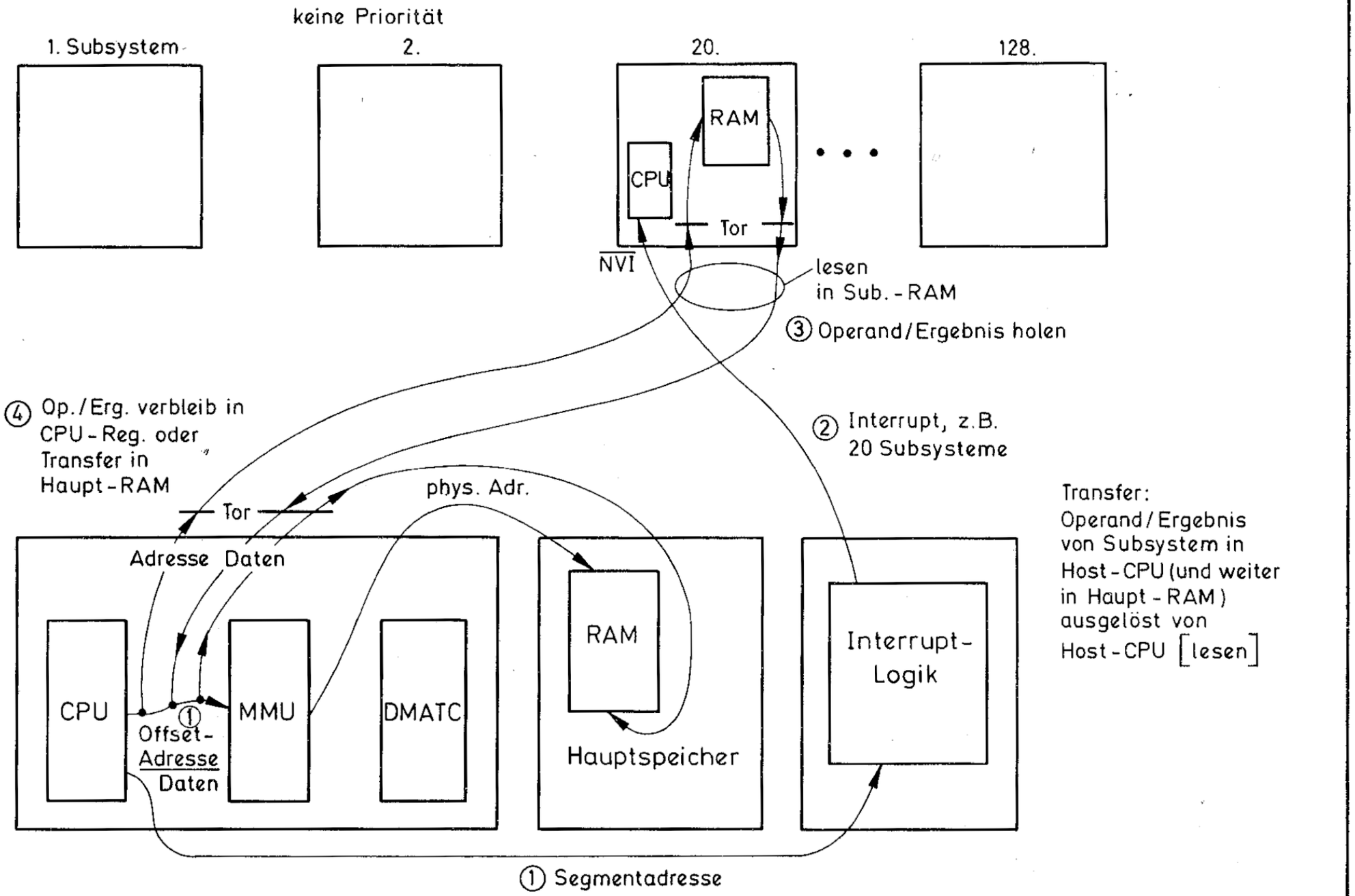

keine Priorität
1. Subsystem
2.
20.
128.
RAM
CPU
Tor
NVI
lesen
in Sub.-RAM
③ Operand/Ergebnis holen
④ Op./Erg. verbleib in
CPU-Reg. oder
Transfer in
Haupt-RAM
② Interrupt, z.B.
20 Subsysteme
phys. Adr.
Tor
Adresse
Daten
CPU
① Offset-
Adresse
Daten
MMU
DMATC
RAM
Hauptspeicher
Interrupt-
Logik
Transfer:
Operand/Ergebnis
von Subsystem in
Host-CPU (und weiter
in Haupt-RAM)
ausgelöst von
Host-CPU [lesen]
① Segmentadresse

Bild 7.10

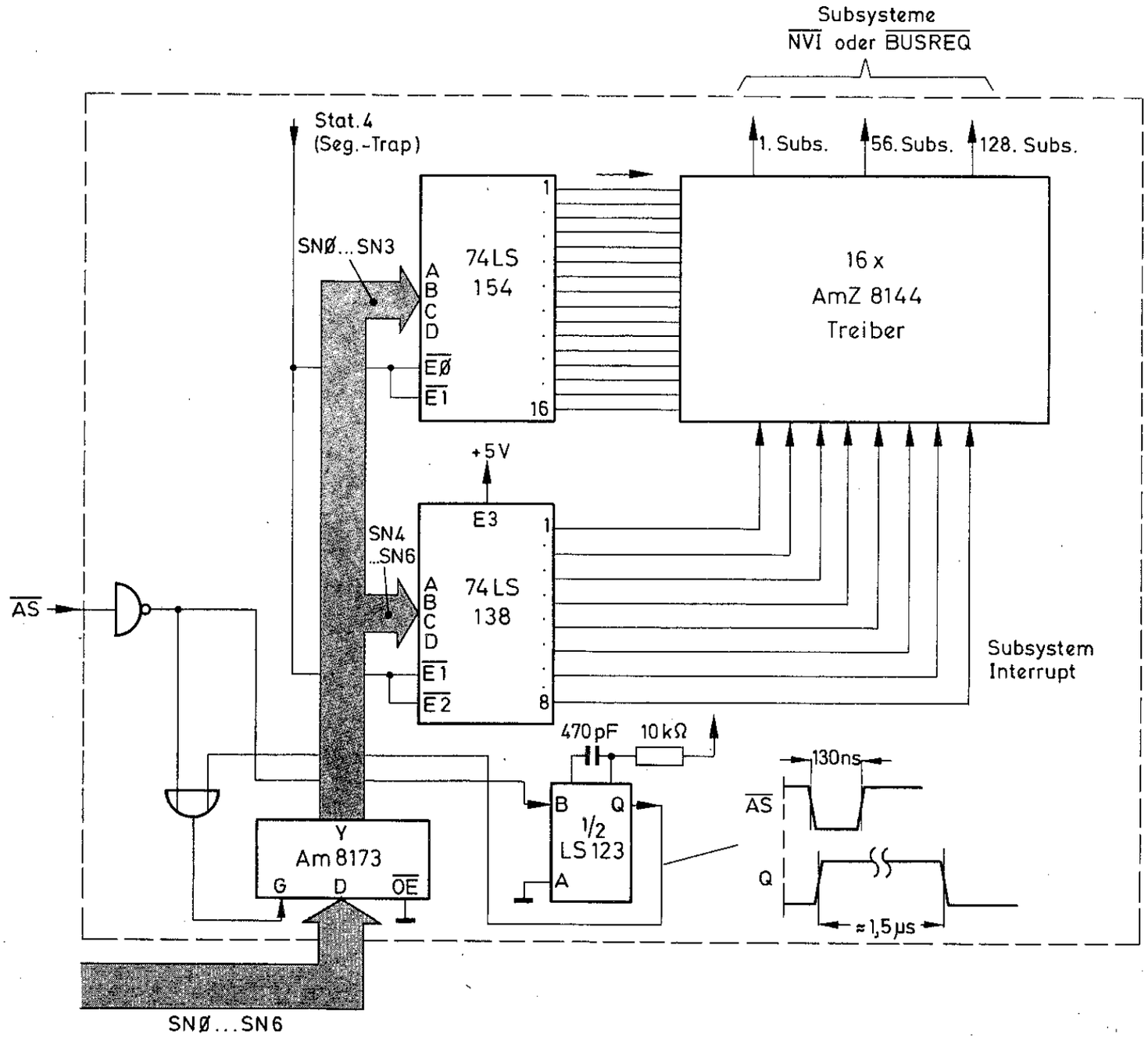

Subsysteme
NVI oder BUSREQ
Stat.4
(Seg.-Trap)
1. Subs.
56. Subs.
128. Subs.
SNØ...SN3
74LS
154
A
B
C
D
EØ
E1
1
16
16 x
AmZ 8144
Treiber
+5V
E3
SN4
...SN6
74LS
138
E1
E2
8
Subsystem
Interrupt
AS
470pF
10kΩ
130ns
B
Q
1/2
LS123
A
AS
Q
≈1,5µs
Y
Am 8173
G
D
OE
SNØ...SN6

Bild 7.11

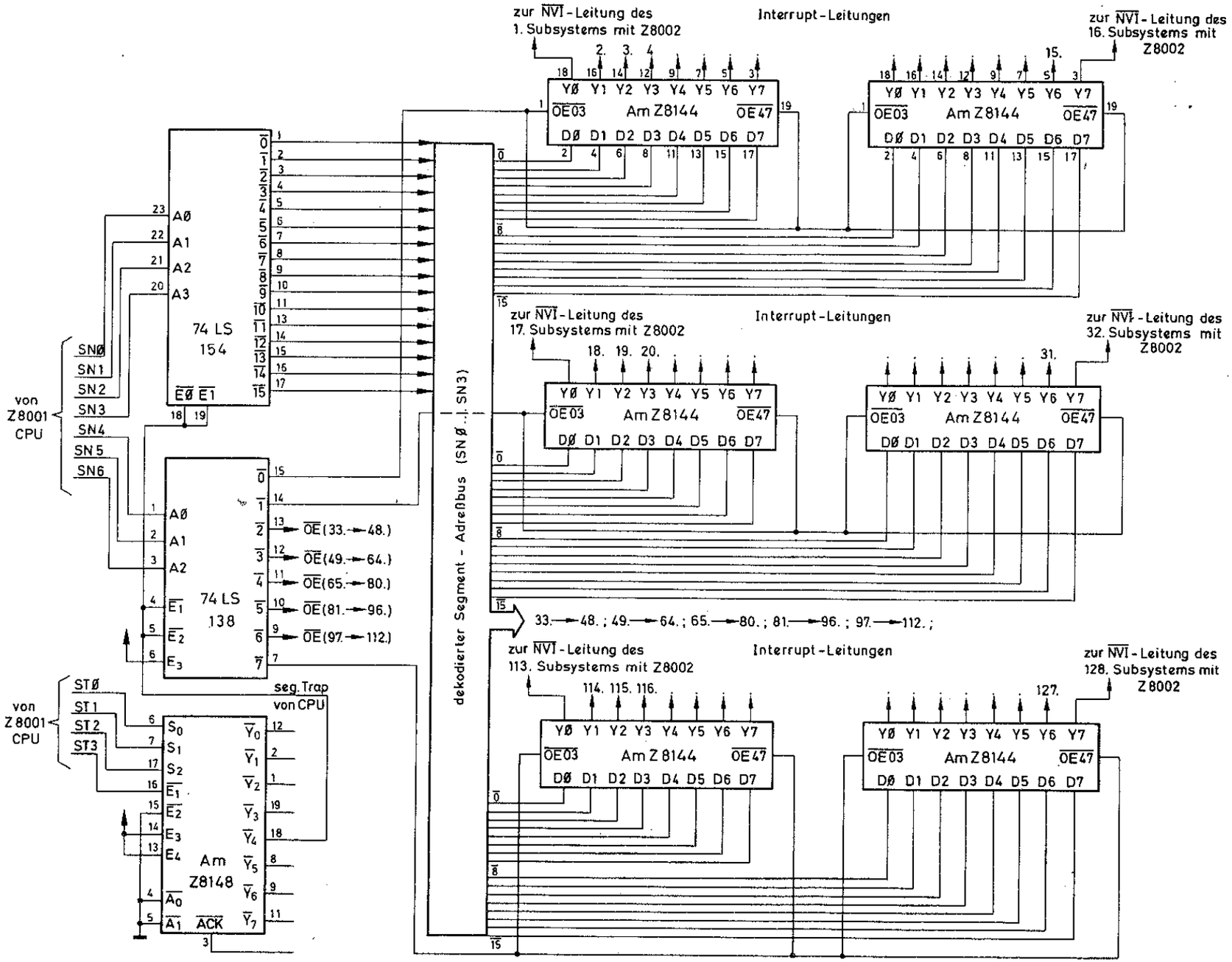

zur NVI-Leitung des 1. Subsystems mit Z8002
Interrupt-Leitungen
zur NVI-Leitung des 16. Subsystems mit Z8002
zur NVI-Leitung des 17. Subsystems mit Z8002
zur NVI-Leitung des 32. Subsystems mit Z8002
zur NVI-Leitung des 113. Subsystems mit Z8002
zur NVI-Leitung des 128. Subsystems mit Z8002
Am Z8144
74 LS 154
74 LS 138
Am Z8148
von Z8001 CPU
seg. Trap von CPU
dekodierter Segment-Adreßbus (SNØ..SN3)
33.→48.; 49.→64.; 65.→80.; 81.→96.; 97.→112.;

Bild 7.12

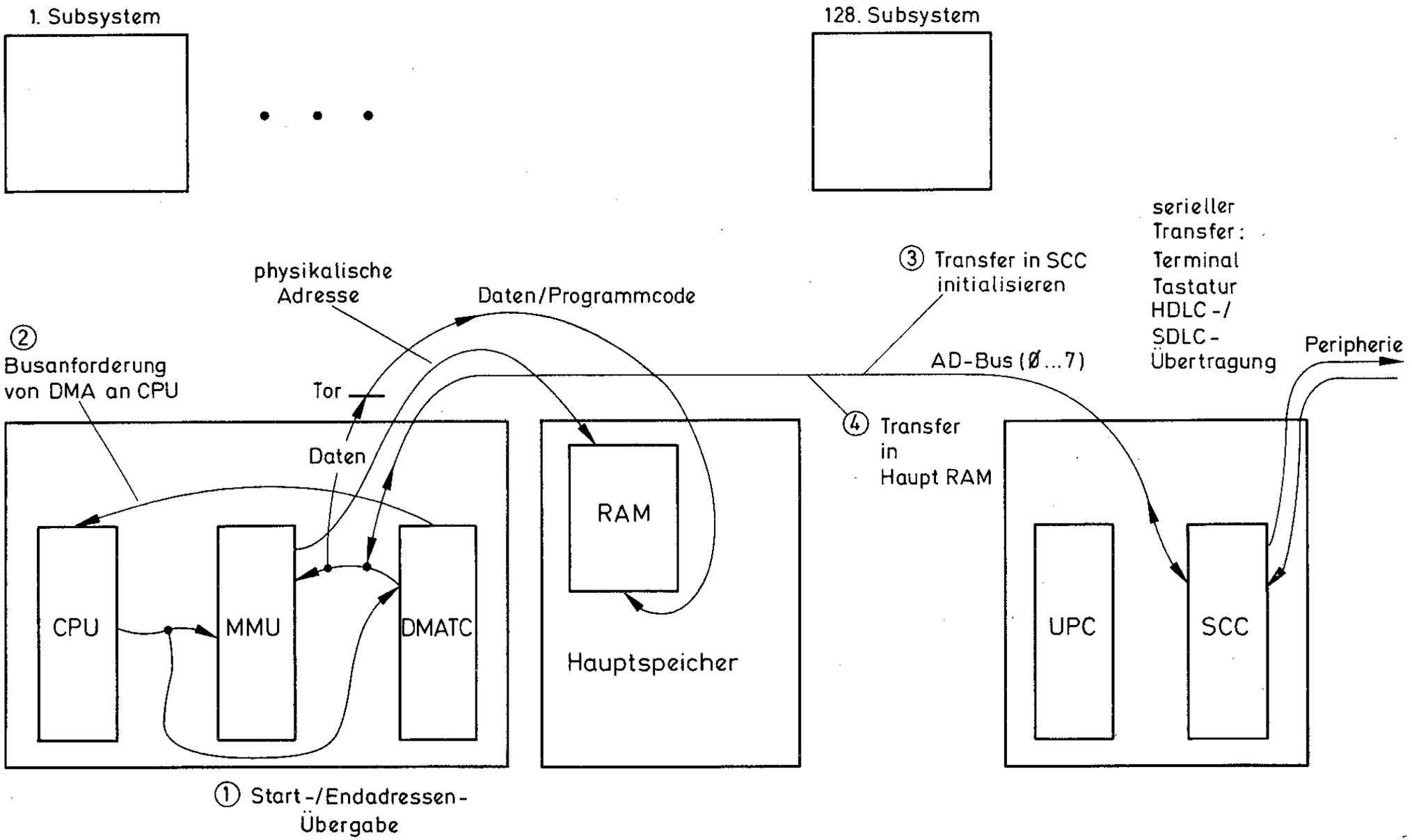

1. Subsystem
128. Subsystem
physikalische Adresse
Daten/Programmcode
③ Transfer in SCC initialisieren
serieller Transfer: Terminal Tastatur HDLC-/ SDLC-Übertragung
Peripherie
② Busanforderung von DMA an CPU
Tor
AD-Bus (Ø...7)
④ Transfer in Haupt RAM
Daten
RAM
CPU
MMU
DMATC
Hauptspeicher
UPC
SCC
① Start-/Endadressen-Übergabe

Bild 7.13

erreicht worden, wird die sequentielle Interrupterzeugung von DONE des 2940 unterbrochen. Auf diese Weise werden eine bestimmte Zahl von $\overline{NVI}$s erzeugt, die die gleiche Zahl von Subsystemen zu einer Host-Computer-Anforderung anregen. Damit ist die Initialisierungsphase eingeleitet.

Die letzte Baugruppe des Parallelrechners betrifft die Ein-/Ausganbebaugruppe mit einer parallelen und seriellen Universalschnittstelle in Form zweier Bausteine der Z 8000-Familie. In **Bild 7.13** ist in schematisch einfacher Form dargestellt, wie der Transfer verläuft. Die Übertragung betrifft ein Terminal, wie auch die in den Massenspeicher (Hard-Disk-Laufwerke). Sie betrifft als Quelle oder Ziel im Rechner ausschließlich den RAM-Abschnitt des Hauptspeichers. Es wird hier darauf verzichtet, auf die genaueren Einzelheiten einzugehen, weil diese Operation einen allgemeinen Charakter besitzt. Davon wird ein Parallelrechner, wie auch jeder andere Rechner betroffen.

In **Bild 7.14** ist die Baugruppe dargestellt. Sie besteht aus der universellen Parallelschnittstelle Z 8034 UPC und dem seriellen Transfer-Baustein Z 8030 SCC, die über das untere Byte des Z 8000-AD-Bus an den Host-Computer angeschlossen sind. Sie sind unmittelbar passende und leicht anzuschließende Systembausteine, die im priorisierten Interruptmodus die Host-CPU anfordern können; Als Interruptebene wird der vektorisierte Interrupt $\overline{VI}$ verwendet. Hierzu gibt der jeweils betroffene Schnittstellenbaustein einen 8-bit-Vektor an die Host-CPU aus, damit sie leicht erkennen kann, woher die Anforderung stammt. Dies wurde bereits in Abschnitt 2.9 genauer betrachtet.

7.3.3 Steuerung durch Software mit Hardwareunterstützung

Das beschriebene Parallelrechnersystem zeigt für viele spezielle Kenner dieses Bereichs, daß die Hardware auf die Software und umgekehrt zugeschnitten ist. Das Ziel dabei ist, eine extrem komplexe Betriebssoftware von vorneherein erst gar nicht entstehen zu lassen. Jeder Kenner weiß, wie teuer Software ist. Intelligente Hardware kann hier eine Menge Softwareentwicklung sparen helfen. Für einen Parallelrechner gilt es insbesondere einen gut dosierten Kompromiß zu schließen. Der Betrieb des hier vorgestellten Z 8000-Parallelrechners ist auf eine für seine extreme Größenordnung sehr geringe Betriebssoftware ausgerichtet. Es handelt sich um kleine, überschaubare Einzelroutinen.

Grundsätzlich ist bei seiner Betriebssoftware dreierlei Art zu unterscheiden:

- Betriebssoftware für die Initialisierung des Hostsystems, — CPU, MMU, DMATC, UPC, SCC . . .
- Betriebssoftware für den Parallelbetrieb, die sich speziell auf die Initialisierungs- bzw. Datenhol- und Austauschphase bezieht.
- Betriebssoftware für den ausschließlichen Betrieb des Hostsystems als Monosystem, die allgemeingültige Prozeduren jedes üblichen Z 8000-Systems unterschiedlicher Größe enthalten kann.

Man muß schon sehr deutlich trennen, welcher Betrag an Software für den Parallelbetrieb notwendig ist. Geht man noch weiter ins Detail, muß man bezüglich des letzten Punktes differenzieren, welche Software als Standardroutinen in dem Firmwarekomplex eines Subsystems unterzubringen ist und welcher Anteil

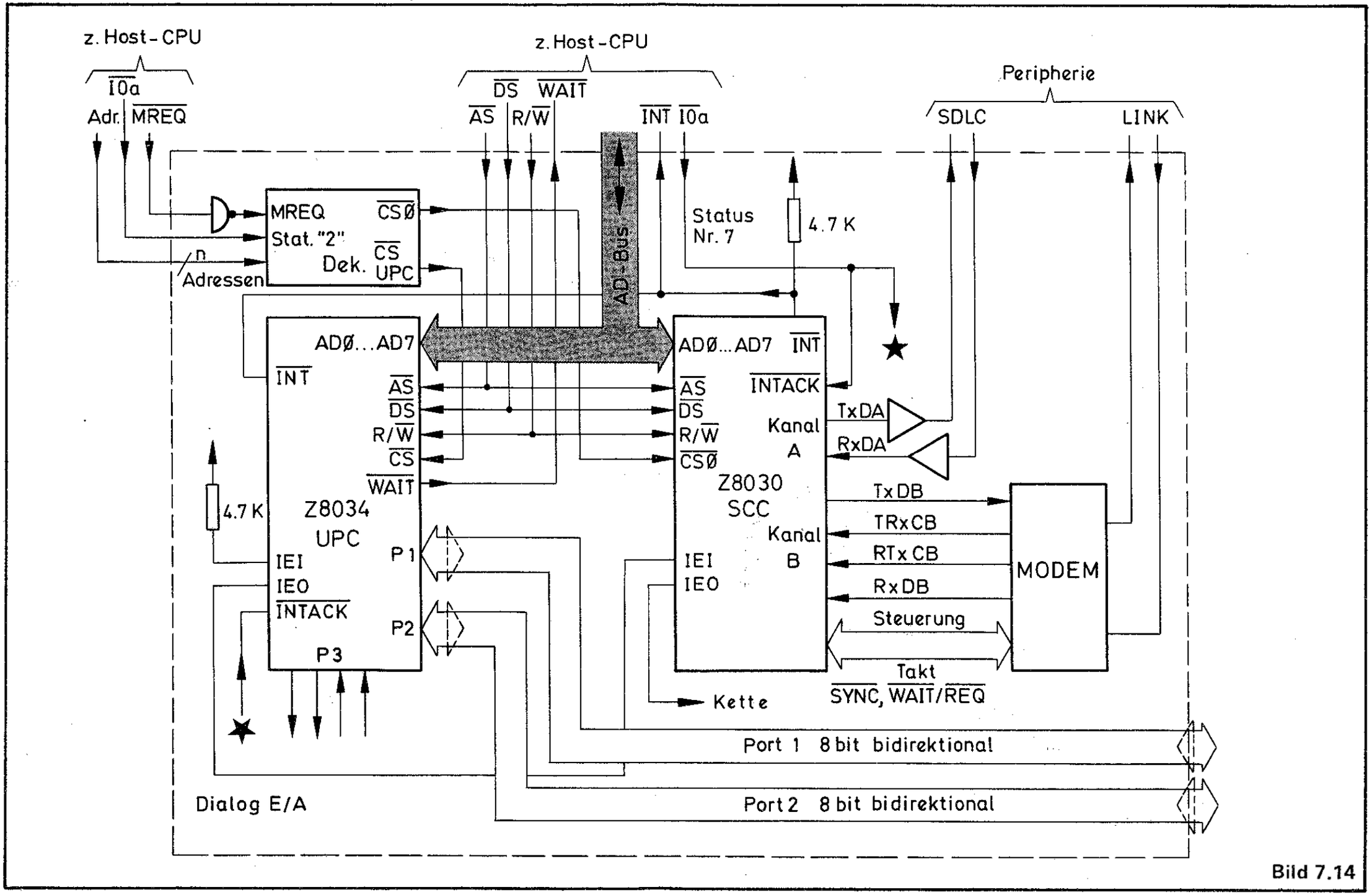

z. Host - CPU
IOa
Adr.
MREQ
z. Host - CPU
DS
WAIT
AS
R/W
INT
IOa
Peripherie
SDLC
LINK
AD-BUS
MREQ
Stat. "2"
Dek.
CSØ
CS
UPC
n
Adressen
Status
Nr. 7
4.7 K
ADØ...AD7
INT
AS
DS
R/W
CS
WAIT
Z8034
UPC
IEI
IEO
INTACK
P1
P2
P3
4.7 K
ADØ...AD7
INT
AS
DS
R/W
CSØ
INTACK
Kanal
A
Kanal
B
Z8030
SCC
IEI
IEO
TxDA
RxDA
TxDB
TRxCB
RTxCB
RxDB
Steuerung
Takt
SYNC, WAIT/REQ
MODEM
Kette
Port 1 8 bit bidirektional
Port 2 8 bit bidirektional
Dialog E/A

Bild 7.14

des Ganzen ausschließlich als Multiprozessor-Software im Host-Computer-Betriebssystem enthalten sein muß.

Eine mehrfach erwähnte Routine betrifft das Subsystem. Wurde ein $\overline{NVI}$ ausgesandt, betrifft das das Durchlaufen eines Multi-Mikro-Anforderungsprotokolls, wie es in Bild 2.60 bereits dargestellt wurde. Diese Routine ist ein Programm, das zur Parallelrechner-Betriebssoftware gehört. Daß die Routine im Rahmen eines Interrupt behandelt wird, bezieht sich auf die "Slave"-Struktur des Rechners. Grundlos fordert ein Subsystem den Host-Computer nicht an. Ist eine Teil-Task-Routine einer Anwenderprozedur von einem Subsystem abgeschlossen worden, zeigt sich die Fertigmeldung wieder darin, daß das Multi-Mikro-Anforderungsprotokoll bearbeitet wird. Der Host-CPU wird angezeigt, daß sie sich die Ergebnisse abholen kann. Das Multi-Mikro-Anforderungsprotokoll hat für einen Parallelrechner der beschriebenen Art **die** elementare Bedeutung. Mit dieser Routine (in jedem Sub-Firmware-ROM) wird jeglicher Zugriffskonflikt eliminiert.

Die zweite angesprochene Routine ist ebenfalls in das Subsystem einzuordnen. Wird ein $\overline{NMI}$ einer Sub-CPU übergeben, bedeutet das, daß sie die Tore ihres Systems "dicht machen muß", weil entweder der Sub-RAM überzulaufen droht oder ganz einfach der Transfer beendet ist. Führt die Sub-CPU ein Programm aus, ist eine weitere Routine dafür zuständig, daß Ergebnisse immer auf definierte RAM-Plätze verschoben werden. Diese vereinbarten Adressen kennt die Host-CPU, so daß eine Routine-Datenholphase immer auf die gleichen RAM-Plätze aller Subsysteme abgezielt ist.

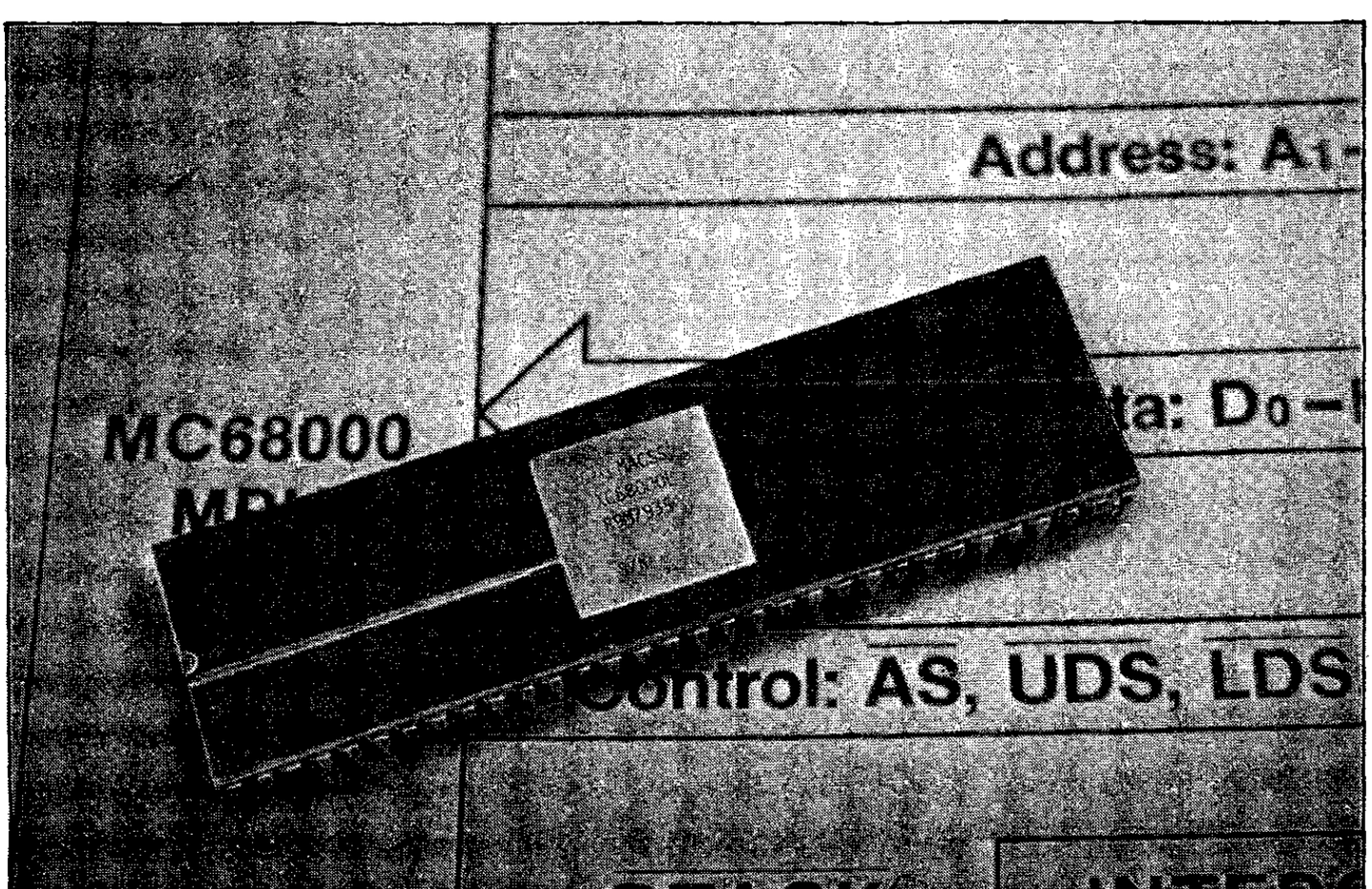
Für die Host-CPU gilt beispielsweise die Einleitung einer Initialisierungsphase als für sie betreffende Multiprozessor-Software. Hier muß gesichert sein, daß die Host-Interrupterzeugung für $\overline{NVIs}$ eine Zahlvorgabe erhält und gestartet wird, Die Host-CPU muß ebenso für die Initialisierung des DMATC sorgen, wenn er das richtige Teil-Task aus dem physikalischen Speicherfeld holen und den jeweiligen Subsystem übergeben soll. Diese Host-Computerroutinen gehören zu einer echten Parallel-Betriebssoftware und sind in das main-frame-Betriebssystem einzuordnen.

Es gibt noch eine Reihe ungenannter kleiner Betriebsroutinen. Generell ist zu diesem Z 8000-Parallelrechner noch einiges zu bemerken. Ein Einstieg in noch speziellere Details würde den Rahmen in dieser Dokumentation sprengen. Dieses Multiprozessorsystem zeigt die künftigen Rechnerkonzepte mit Mikrocomputer-Bausteinen. Sie setzen damit ihre obere Grenze. Diese obere Grenze heißt "Größtrechner".

Z8000 – und wer noch?

8

Diese Überschrift als "Statement" betrachtet, läßt auf einen Eindruck ziemlicher Überheblickeit schließen. Die Überschrift soll hingegen auf den tatsächlich exklusiven Kreis hindeuten, in dem sich der Z 8000 befindet. Realität zu Beginn der 80er Jahre ist, daß neben dem Zilog-Baustein der TMS 9900 von Texas Instruments (auch von Intermetall ITT), der 8086 von Intel (auch von Mostek und Siemens) und schließlich der 68000 von Motorola (auch von Rockwell) diesem exklusiven Kreis von 16-bit-Prozessoren angehören. Auf der "electronica 80" in München wurde eine Information über das 16-bit-Produkt von National Semiconductor gegeben, den NS 16032, der auch von Fairchild produziert werden wird. Der NS 16032 von National Semiconductor wird in einem strukturreduzierten NMOS-Prozeß, genannt XMOS, mit 3,0 Micron Gate-Arrays entworfen. Sieht man jedoch davon ab, daß dieses Quasi-32-bit-Produkt 1980 noch nicht existierte, bleiben drei große 16er übrig, die die obere Kategorie bilden, die Typen 8086, Z 8000 und 68000. Dieses Trio muß, genaugenommen, nochmals reduziert werden. Obwohl die Firma Intel große Anstrengungen unternimmt (z.B. 8-MHz-Version des 8086), wird die Leistung des Z 8000 in den einzelnen Kriterien nicht erreicht. Die 8086-Familie wurde mit dem E/A-Prozessor 8089 weiter ausgebaut. Dieser Baustein arbeitet nach dem sehr interessanten und zukunftsträchtigen Prinzip der direkten Busbeobachtung. Das Prinzip deckt sich mit dem der Zilog-EPA. Die Z 8000-CPU und die EPA (EPU) sind in ihrer Zusammenarbeit (Transferfunktion) der Kombination 8086 und 8089 sehr ähnlich. Die Intel-Kombination CPU-E/A-Prozessor



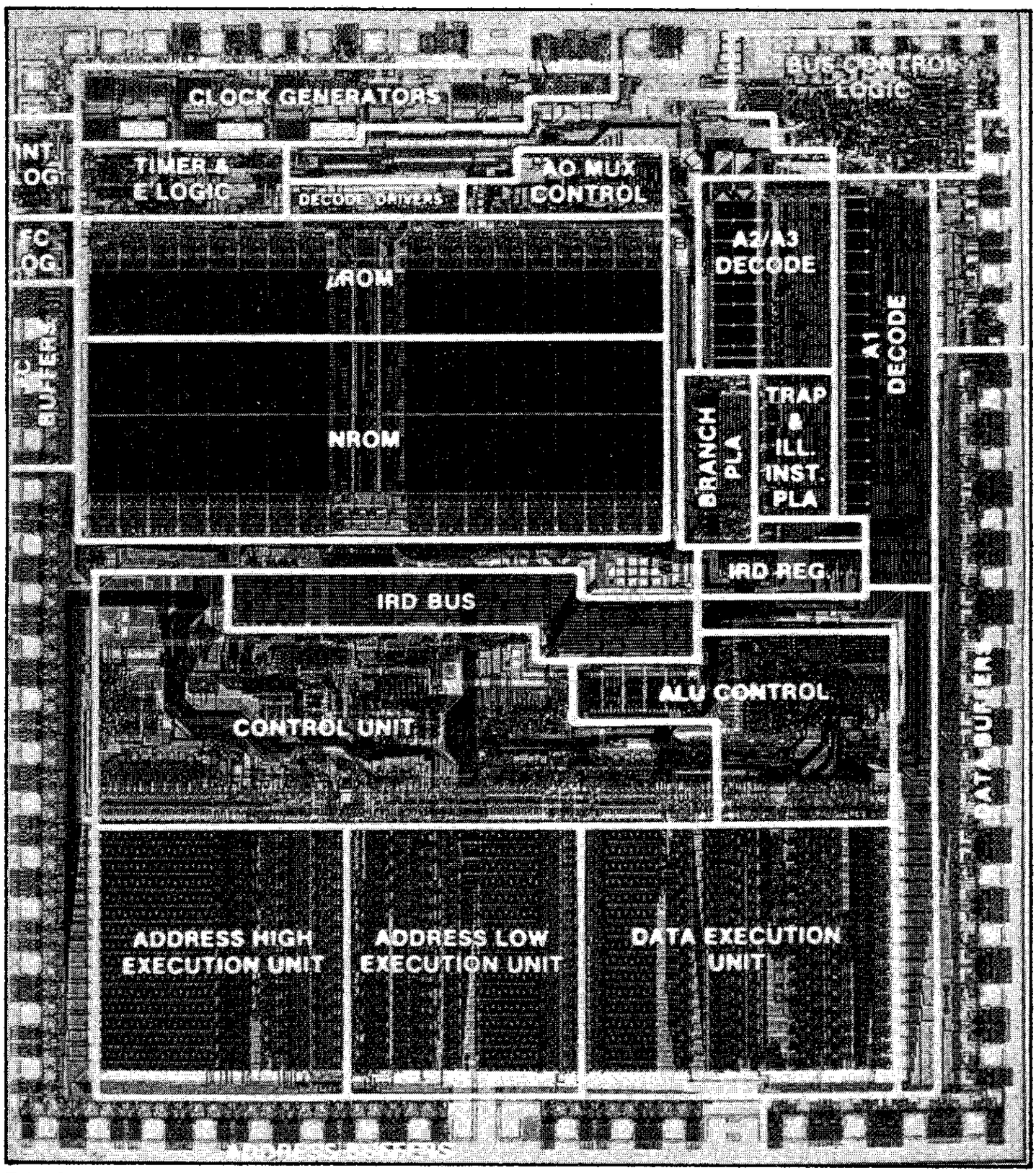


Bild 8.1

bewirkte zweifellos eine Aufwertung der 8086-CPU. Das ändert trotzdem nichts an dem Leistungsgefälle zu Ungunsten der Intel-CPU. Begrenzt man die Betrachtung ausschließlich auf die CPUs, stellen sich zu Beginn der 80er Jahre als die ernsthaft zu bezeichnenden Extreme der 16-bit-Szene zwei Bausteine heraus:

Zilog's und AMD's Z 8000 gegenüber **Motorola's 68000.** Der bisherige Inhalt dieses Buches zeigte sehr anschaulich die enormen Stärken der Z 8000-CPU. Im folgenden Abschnitt wird ein kurzer, nüchterner Vergleich dieser beiden "Pole-Positions" angestellt, der sicherlich mit einiger Brisanz versehen ist.

8.1 Z 8000 – 6800, EINE KLEINE ANALYSE

Die 68000-CPU wird in dem strukturreduzierten "High Density Kurzkanal-NMOS-Prozeß" hergestellt. Motorola bezeichnet ihn als HMOS I. Eine elementare Zellgeometrie wird darin mit 3 . . . 4 Micron definiert (Mikrometer). Die absolute Fläche des 68000-Chip beträgt 246 mal 281 mil² = 69,1 Kmil². Die Länge 1 mil entspricht 0,001 inch (= 0,0254 mm). Der 68000 ist folglich

44,597 mm² groß. Die Chipfläche des Z 8000 beträgt 60,9 Kmil² = 39,3 mm². Bezieht man die Zellgeometrie beider Technologien zueinander, ergibt sich eine Flächenverkleinerung von Z 8000 zu 68000 wie ≈ 2,5:1 (MOS-Transistor Z 8000 zu 68000 MOS-Transistor). Der Chip der 68000-CPU beinhaltet etwa 70.000 Transistoren oder 13.000 Gatter. Bemerkenswert an diesem Chip **(Bild**

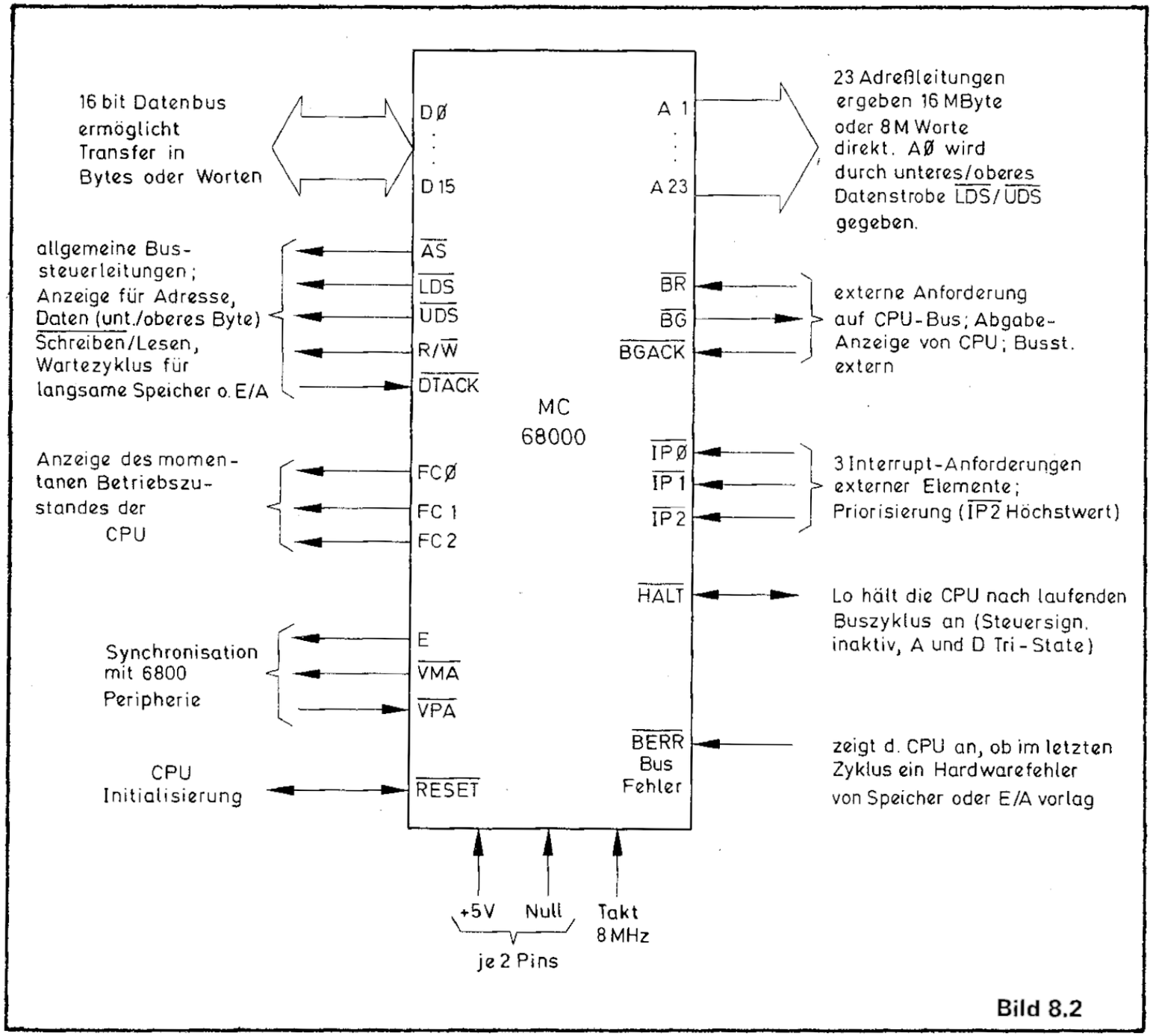


Bild 8.2

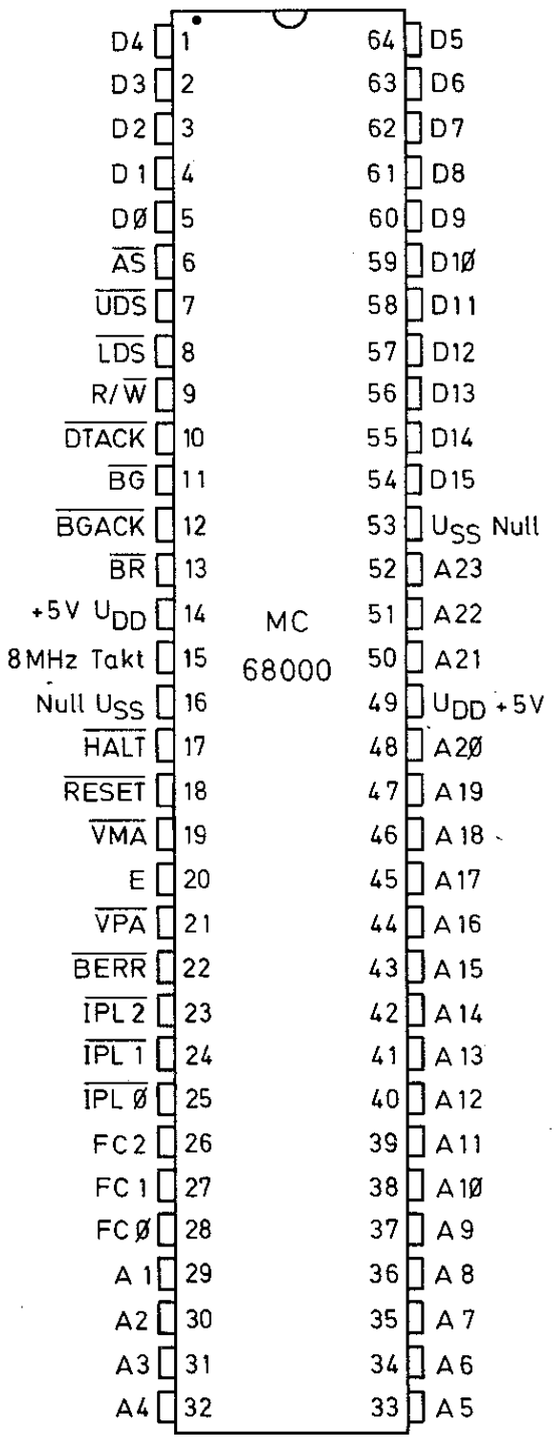


Bild 8.3

8.1) ist, daß die CPU bei einem Takt von 8 MHz nur 1 W Leistung verbraucht (Z 8000A – 6 MHz – etwa 1,5 W). Würde der Z 8000 im identischen Prozeß hergestellt, wie er für den 68000 angewandt wird, wäre die Z 8000-Chipfläche nur 34 Kmil^2 (= 24 mm^2) groß.

Der wesentliche Unterschied in der logischen Struktur dieser beiden Extreme liegt in der Befehlsdekodierung. Der 68000 hat hierfür einen eigenen mikroprogrammierten Computer auf dem Chip, der für die Ausgabe und die richtige Sequenz der Steuersignale sorgt, damit ein Befehl ordnungsgemäß abläuft. Tatsächlich ist dies der einzig gangbare Weg, eine solch gewaltige Logik, wie sie der 68000 nun mal besitzt, ordnungsgemäß zu koordinieren. Darin liegt der extrem hohe Bedarf an Gattern begründet (etwa 3 mal so viele, wie beim Z 8000). Die eigentliche Befehlsausführung des 68000 ist, physikalisch betrachtet, langsamer als beim Z 8000. Eine optimierte, feste Logik mit kürzesten Wegen und Laufzeiten (Z 8000) ist physikalisch schneller. Bei Motorola weiß man das selbstverständlich genauso wie bei Zilog. Der Vorteil des 68000 liegt eindeutig darin, daß sein Befehlssatz relativ variabel geändert werden kann. Dazu ist prinzipiell nur das Mikroprogramm für die Mikrobefehlssequenz eines Makros zu ändern. Eine Korrektur des Befehlssatzes (Makros) ist beim Z 8000 nicht so einfach durchführbar. Beim Z 8000 ist der Befehlssatz gleichsam verdrahtet, wogegen beim 68000 ein Mikroprogramm dafür steht. Folglich sind beide CPUs logisch elementar verschieden. Bei der Anwendung merkt man davon praktisch nichts. Wäre die Z 8000-Logik in der gleichen Technologie wie der 68000 erstellt, der Z 8000 wäre zwangsläufig ernsthaft schneller in seiner Befehlsausführung und damit in fast allen Benchmarks. Für beide CPUs gilt jedoch: In der jeweiligen Art der Befehlsdekodierung repräsentieren beide eine absolute Meisterleistung.

Vergleicht man beide CPUs bezüglich der Signaldiagramme, fällt bemerkens-

wertes auf. Für den Z 8000 gilt, wie in Kapitel 2 gezeigt wurde, Lese- oder Schreibzyklus grundsätzlich mit 3 T-Zyklen ohne WZ als durchgeführt. Nicht so beim 68000. Ohne WZ benötigt er hierfür immer 4 T-Zyklen (übrigens ebenso der 8086). Der Z 8000 ist somit um einen Takt schneller, obwohl er einen Adreß-/Datenbus besitzt, der im Zeitmultiplex-Betrieb arbeitet. Der 68000 hat hingegen einen sehr interessanten Eingang. Die Anschlußstruktur des 68000 ist in **Bild 8.2** dargestellt. Wird der Eingang "Bus Error" $\overline{\text{BERR}}$ der CPU von irgendeiner externen Busüberwachungslogik auf Lo gesetzt, bricht die CPU den laufenden Befehlszyklus ab. Sie behandelt den Vorfall entweder damit, daß sie nochmals den vorhergehenden Buszyklus (Speicher- oder E/A-Transfer) unter gleichen Bedingungen wiederholt ($\overline{\text{BERR}}$ wurde in einer Holphase aktiviert) oder ein Bus-Ausfallzyklus wird von der CPU ausgegeben. Die Reaktion der CPU ist abhängig vom Zustand des $\overline{\text{HALT}}$-Anschlusses bei $\overline{\text{BERR}}$ gleich Lo. Der Bus-Ausfallzyklus (Initiator einer Art Interrupt-Serviceroutine) wird auf Grund nicht reagierender Speicherbausteine, fehlerhaft übertragener Interruptvektoren oder illegaler Zugriffe auf Speicherbereiche unter Memory-Management-Kontrolle aktiviert (letzterer Grund ist mit einem Z 8000-Segment-Trap vergleichbar). Der Z 8000 besitzt keinen $\overline{\text{BERR}}$-Eingang. Der 68000 löst den Start der "Bus-Ausfall"-Serviceroutine mit einem Adreßsprung in der Ereignistabelle auf den Wert . . . % 008 als Vektor. Er verhält sich in diesem Fall sehr ähnlich wie der Z 8000 mit seiner Sprungtabelle in Bild 2.22. Die Lösung eines Vorfalls der genannten Art ist mit dem Z 8000 entweder in einer Interruptebene oder über den Segment-Trap-Eingang anzugeben.

In Bild 2.28 ist der 68000-Registersatz dargestellt. Hier zeigt sich eine echte Überlegenheit des 68000 gegenüber dem Z 8000. Ist der Z-Registersatz bereits außergewöhnlich, hat der 68000 hier tatsächlich die extremste Registerstruktur aufzuweisen, die jemals in einer CPU monolithisch integriert wurde. 60 Byte Adreß- und Datenregister ist extrem. Jedoch, einen zusätzlichen Biß enthält der Z 8000. Mit seinen Vierfach-Ergebnisregistern (64 bit Länge) für MULT und DIV ist er gegenüber dem 68000 nicht als zweitrangig zu betrachten.

Eine ausgesprochene Stärke zeigt der Z 8000 mit seinen echten 32 bit MULT- und DIV-Befehlen. Sein Befehlssatz zeigt ferner eine außerordentliche Dominanz ausgeprägter Zeichenkettenbefehle (siehe S. 3-45), für die der 68000 kein Äquivalent bietet. Als ebenso einzigartig gelten die vier Multiprozessorbefehle, die mit den beiden CPU-Anschlüssen $\overline{\mu\text{I}}$ bzw. $\overline{\mu\text{O}}$ gekoppelt sind. Beim 68000 ist über $\overline{\text{BR}}$, $\overline{\text{BG}}$ und $\overline{\text{BGACK}}$ eine Busübergabe hardwaremäßig einfach zu gestalten. Die außerordentliche Klasse beider Prozessoren dokumentiert sich in vielfältigen Punkten, wobei beide eine Reihe von Gemeinsamkeiten aufweisen, die bei anderen CPUs einfach nicht zu finden sind.

In der folgenden Tabelle werden die wesentlichen Eigenschaften dieser beiden Superprozessoren gegenübergestellt. Die Anschlußbelegung des 68000-Bausteins zeigt **Bild 8.3.**

Technische Eigenschaften

Z 8000	68000
Packung	
48/40poliges DIL-Gehäuse	64poliges DIL-Gehäuse
Taktzyklus	
250 ns und 166 2/3 ns (A-Typ)	250 ns, 166 2/3 ns und 125 ns
Adressen, Daten	
Gemeinsame Anschlüsse, Zeitmultiplex	Getrennte Anschlüsse
Elementares Signalverhalten, Lesen oder Schreiben in Speicher ohne WZ	
3 T-Zyklen	4 T-Zyklen
Allzweckregister	
16 Register (Akku, Index, Stapel) je 16 bit Länge, inklusive Stapelzeiger System, Normel	8 Datenregister 7 Adreßregister getrennt alle Register mit 32 bit Länge 2 Stapelzeiger separat
Interruptebenen	
nicht maskierter I. $\overline{NMI}$ (verwendbar: Segment Trap $\overline{SEGT}$) Vektorinterrupt VI $\overline{VI}$ (256 8002, 128 8001) nicht vektorisierter I. $\overline{NVI}$	$\overline{IPL\ 2}$ höchstwertiger I. $\overline{IPL\ 1}$ $\overline{IPL\ 0}$ niederwertigster I. $\overline{BERR}$ Bus-Fehler (Ausfallzyklus)
Adreß-Kapazität	
64 KB ein Segment Z 8002 128 x 64 KB (= 8 MB) Z 8001	24 Adreßleitungen, A1 . . . A23, DDS, UDS 16 MB segmentfrei
Datentypen	
Einzelbits, BCD Digits, Bytes (8 bits), Worte (16 bits), Doppelworte (32 bits), Byte-Strings, Wort-Strings	Einzelbits, BCD Digits, Bytes, Worte, Doppelworte
Adressierungsarten	
8 Grundtypen: Register direkt Register indirekt unmittelbare Adressierung (dir. Op.) direkte Adressierung indizierte Adressierung relative Adressierung Basis-Adressierung Basis indizierte Adressierung	6 Grundtypen: Register direkt Register indirekt unmittelbare Adressierung (dir. Op.) direkte Adressierung relative Adressierung implizite Adressierung

Technische Eigenschaften

Z 8000	68000
Befehlssatz	
9 Befehlsgruppen 110 Befehlstypen 414 Einzelbefehle	7 Befehlsgruppen 56 Befehlstypen vielfältige Variationen
Besondere Merkmale im Befehlssatz	
16 bit MUL 70 . . . 72 T-Zyklen 16 bit DIV max. 97 T-Zyklen 32 bit MULTL 284 + 7 · n T-Zykl. 32 bit DIVL 725 T-Zyklen	16 bit MULU, MULS 70 T-Zyklen 16 bit DIVU, DIVS 140, 158 – –
Außerordentlich leistungsfähige Block-Transfer- und String-Operationen	–
4 Befehle zur Synchronisationshilfe in Multiprozessor-Netzwerken	–
Korrektur vorhergehender binärer Addition oder Subtraktionen mit dem Befehl DAB	Binär kodierte, reine Dezimalrechnung direkt ausführbar: Addition mit ABCD, Subtraktion mit SBCD und Negation mit NBCD
Bitmanipulation mit den Befehlen BIT, RES, SET, TEST	Bitmanipulation mit den Befehlen BCHG, BCLR, BSET, BTST
Schneller Transfer von Register-inhalte mit LDM	Schneller Transfer von Register-inhalte mit MOVEM
Traps	
bei nicht implementierten Befehlen, bei der Ausführung eines privilegierten Befehls im Normal-Modus, bei einem Befehl, der die EPA betrifft (EPA-bit im FCW nicht gesetzt) bei einer Verletzung einer Zugriffs-definition eines Segments (MMU)	bei nicht implementierten Befehlen, bei der Ausführung eines privilegierten Befehls wie STOP, RESET, RTE, MOVE to SR . . .
Sonstiges	
Sehr gute Vorbereitungen im Befehls-satz der CPU bei Initialisierungs-routinen mehrerer MMUs in einem System.	Raffinierte Fehlererkennung von Softwarebugs: bei Wortzugriff mit ungerader Adresse, bei fehlerhaftem Speicherbus bzw. Transfer, bei einer Division durch Null, bei einem Überlauf-Bedingungscode (TRAPV-Befehl), Trap bei einer Überschreitung der Operandenlänge in einem Register (CHK-Befehl nach Trap)

Bestehende Z8000-Systeme

9

Die laufende Entwicklung immer komplexerer Bausteine mit mehr und mehr Leistungsfähigkeit haben die Hersteller zunehmend zu Forschungsinstitute der Computertechnik avanciert. Die Computerbausteine bestimmen die Eigenschaften des Computers, vielweniger die Netzwerke, die Bausteine zu einem System werden lassen. Folglich erwarten heute die Anwender dieser Technologie, daß die Hersteller dieser LSI- und VLSI-Bausteine ebenso Hersteller einer neuen Computergeneration mit diesen Bausteinen sind. Ein Halbleiterhersteller muß gleichsam ein **"Systemhaus"** sein. Diese Erkenntnis ist bereits zur Leitlinie im Management einzelner Firmen geworden. Somit umfaßt das Angebot neben den Bausteinen ein Platinensystem (CPU, Speicher, E/A, Diskettensteuerung usw.) der jeweiligen Bausteinfamilie bis hin zu kompletten Computersystemen. Abgerundet wird das Programm mit grundlegender, sowie zugeschnittener Software, insbesondere Betriebssysteme. Nach diesem ungeschriebenen Gesetz bezüglich eines Komplettangebots "Hardwaresysteme plus Betriebssoftware" ist es folglich verständlich, daß Zilog wie auch AMD die ersten Z 8000-Systeme anboten.

9.1 Z 8000 ENTWICKLUNGSMODUL VON ZILOG

Das erste funktionierende Z 8000-System wurde von der Firma **Zilog** mit der Originalbezeichnung "Z 8000 Development Module" angeboten. Man verfolgte damit die Zielsetzung, ein vollständiges Software-Entwicklungssystem anzubieten. Ein Betrieb dieses Einplatinen-Computers stützt sich auf das bereits eingeführte Z 80-Hostsystem mit der Bezeichnung PDS 8000. Die Betriebsbasis bildet das Disk-Operating-System RIO, das bei Zilog entwickelt wurde. Dieses Betriebssystem repräsentiert die Basissoftware der MCZ- sowie ZDS-Systeme. Das sind relativ groß ausgebaute Mikrocomputersysteme mit der Z 80-CPU. Die Vollständigkeit des Z 8000 DM-Boards zeigt sich in der Zweifach-Serienschnittstelle nach dem RS 232 C-Format, das mit dem Baustein Z 80A SIO/2 realisiert ist. Damit wird eine Standardverbindung zum Hostsystem PDS 8000 und zum Bildschirmterminal hergestellt. Das Z 8000 DM-Board hat ein eigenes kleines Monitorprogramm (resident in zwei EPROMs des Typs 2716), das sowohl den Monitor-Modus und den Transparent-Modus gestattet. Im Monitor-Modus wird direkt mit dem Board korrespondiert. Die Befehle des Monitors ermöglichen das Arbeiten mit dem Board. Der Transparent-Modus wird aktiviert, wenn man über das Terminal mit dem Z 80-Hostcomputer arbeiten will. In diesem Fall befindet man sich über CR der Tastatur im RIO-Betriebssystem des PDS-Systems. Das Monitorprogramm des DM-Boards stellt eine sinnvolle Erweiterung zum RIO dar und sollte deshalb nicht isoliert gesehen werden. Andererseits besitzt es gerade so viel Eigenständigkeit, daß man den DM-Computer über ein Terminal (CRT plus ASCII, ohne Host-Computer) über eine RS 232 C-Leitung bedienen kann, sprich, im HEX-Code programmieren kann.

In **Bild 9.1** ist das Z 8000 DM-Board dargestellt. Neben der Steckerleiste zum Anschluß der Stromversorgung (ganz

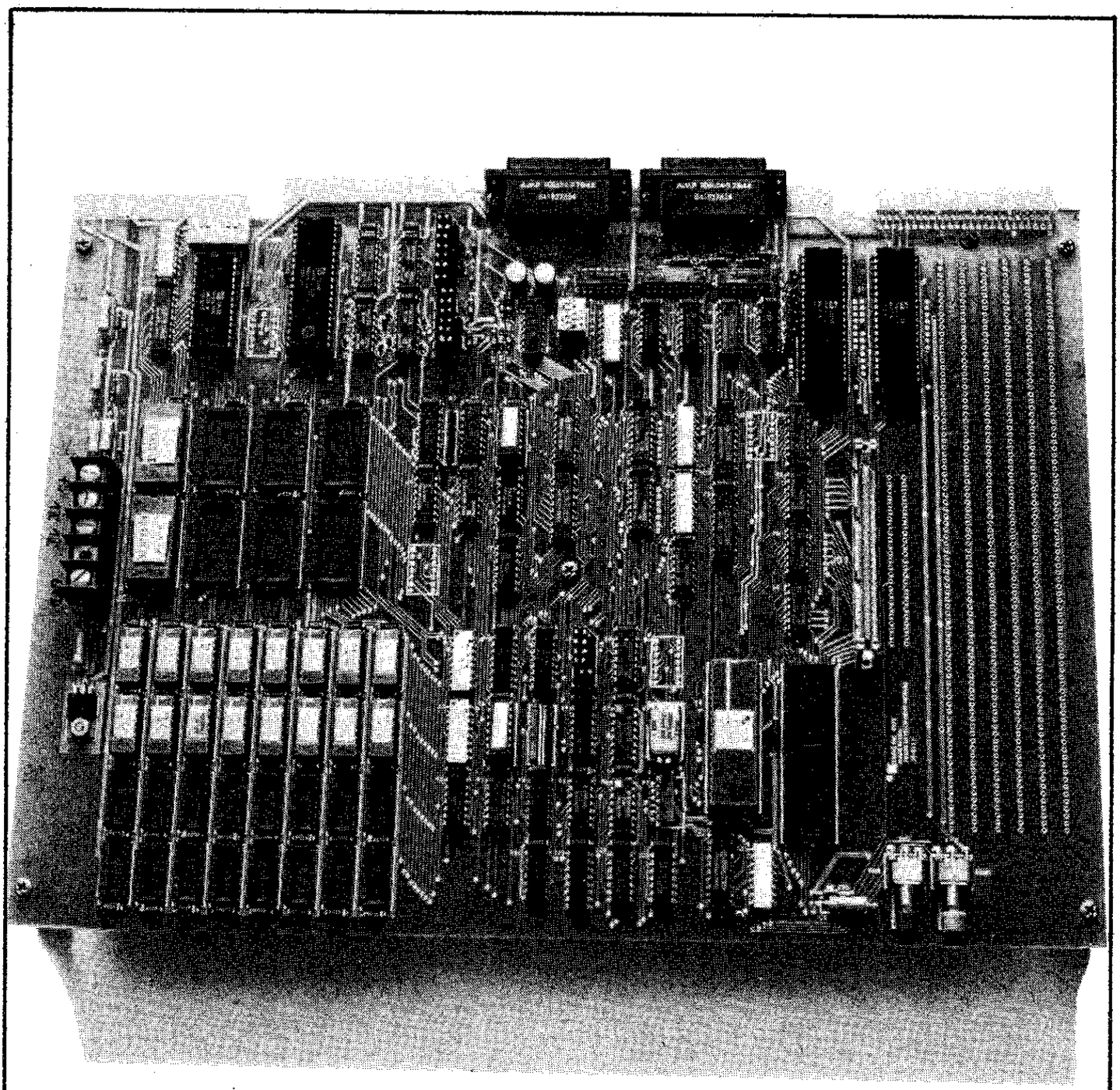
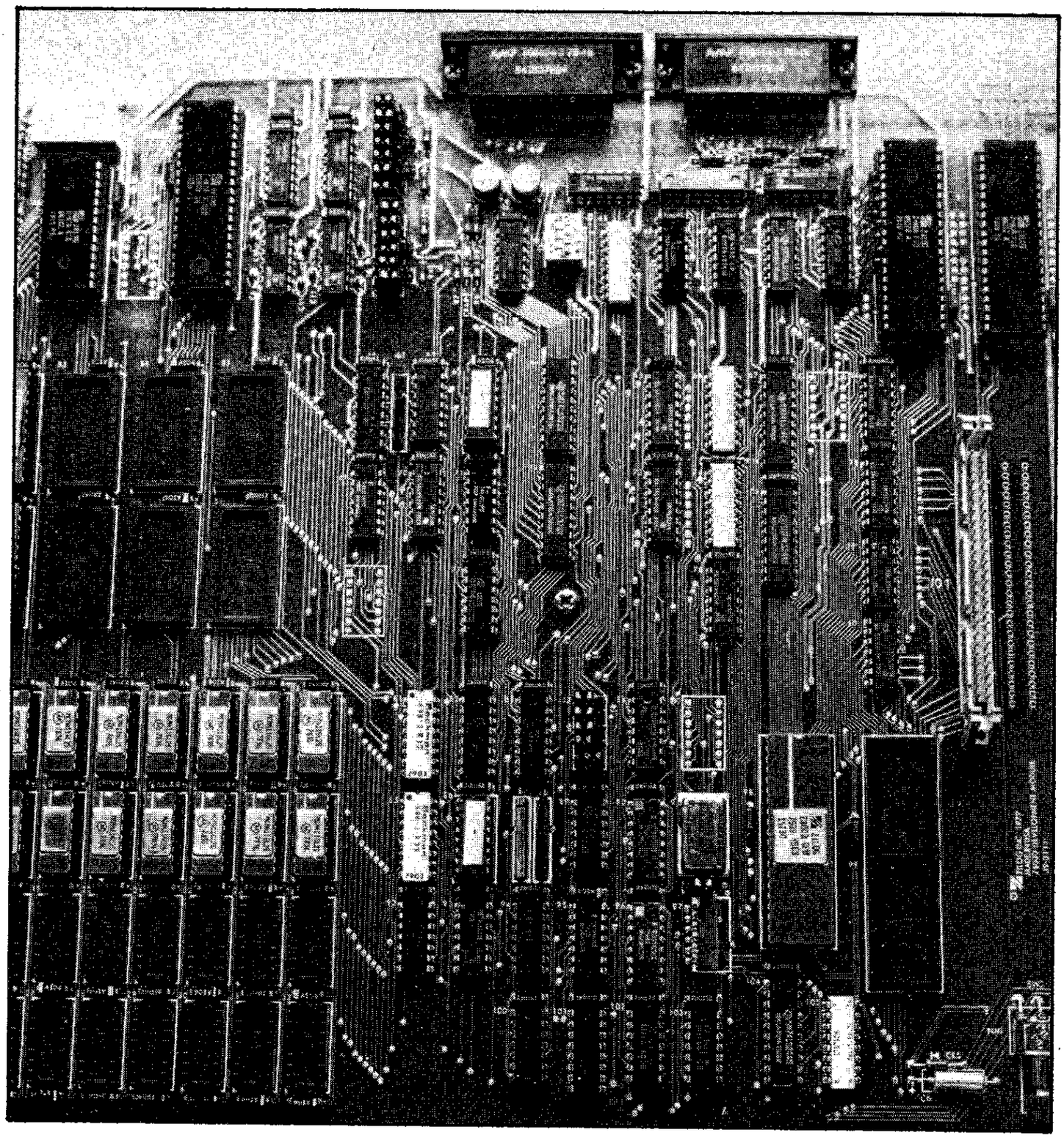


Bild 9.1

links) liegt das Feld der Festwertspeicher, die mit dem Monitor in zwei von acht Fassungen belegt sind. Unterhalb ist das Feld des Arbeitsspeichers gelegt, der mit dynamischen RAMs des Typs 4116/6116 in der Größe von 16 K Worte belegt ist. Wie zu sehen ist, wäre der Arbeitsspeicher auf 32 K Worte ausbaufähig (bestehende Fassungen). Die prinzipielle Ansteuerung dieses Speicherfeldes ist hardwaretechnisch nach Bild 4.32 ausgeführt. Für Festwert- und Arbeitsspeicher gilt Byte- und Wortzugriff. Unterhalb rechts sind zwei Fassungen für die CPUs angeordnet, von denen generell eine mit der Z 8002-CPU belegt ist. Die Umrüstung auf den Z 8001 ist möglich, erfordert jedoch gewisse Änderungen im Monitorprogramm und ein wenig Umstecken in der Dekodierung des Speicherwerks. Zur Initialisierung des Boards ist rechts daneben ein Druckknopf zur Auslösung eines $\overline{\text{NMI}}$ und einer für einen RESET angebracht. **Bild 9.2** zeigt den Abschnitt auf dem Board etwas deutlicher. Ferner ist das Board mit Z 80-Peripheriebausteinen ausgestattet. Der wichtigste wurde bereits genannt. Der Baustein Z 80A SIO/2 sorgt für den seriellen Datentransfer über zwei



unabhängige Datenkanäle mit Voll-Duplex-Betrieb. Eine Übertragungsrate von 110 . . . 19200 Baud ist einstellbar. Die Schnittstelle ist sowohl in der Norm RS 232 C oder mit einer 20 mA-Stromschleife für einen Teletype-Anschluß ausgeführt. Genormte Steckverbinder sind vorhanden.

Für einen parallelen Datentransfer stehen 32 E/A-Leitungen zur Verfügung, die durch zwei Z 80A PIO gegeben sind. Für spezifisch zeitlich zu koordinierende Programmieraufgaben steht ein Zähler/Zeitgeber aus der Z 80-Familie zur Verfügung. Dieser Baustein, der Z 80A CTC enthält vier unabhängige Zählerkanäle mit 256 bit Länge. Die vielfältige Programmierbarkeit ermöglicht eine einfache Lösung von Zeitschleifen in Programmen.

Damit sind die Hardwareelemente des Z 8000 DM-Boards kurz beschrieben.

Die Verbindung in ein Zilog-Entwicklungssystem (MCZ, ZDS) ist in **Bild 9.3** einfach dargestellt. Hier ist das Hostsystem mit PDS 8000 bezeichnet. Auf dieser Verbindung basierend, sind speziell zwei Befehle des DM-Monitors abgestimmt.

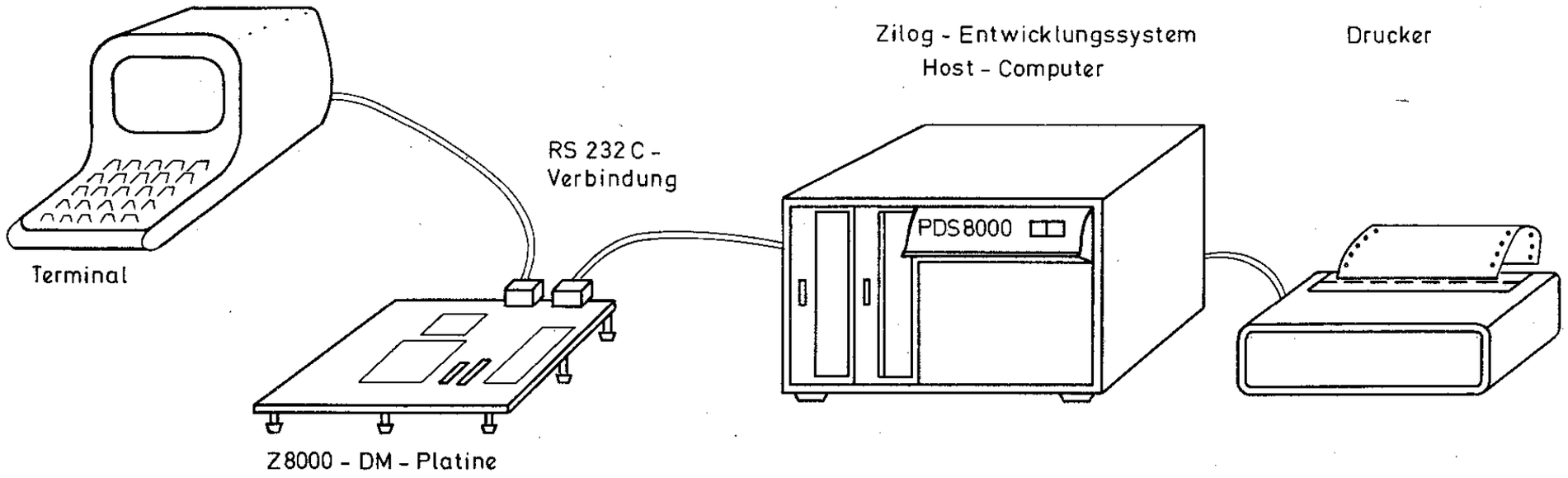


Bild 9.3

LOAD → überträgt Daten vom MCZ-/ZDS-System in das RAM-Speicherwerk des DM-Boards

SEND → überträgt Daten vom RAM-Speicherwerk des DM-Boards in das MCZ-/ZDS-System

Diese beiden Befehle werden im Monitor-Modus eingeleitet. Man drückt die $\overline{\text{NMI}}$-Taste auf dem Board. Es erscheint das Zeichen "<" und man kann daraufhin folgende Eingabe ausführen:

< LOAD (Programmname) CR

z.B. < LOAD UEBUNG 1 CR.

Das Z 8000-DM befindet sich in diesem Fall im Transfer-Modus, das zugleich auch eine Ausführung eines Z 80-Programms im MCZ-/ZDS-System bewirkt. Es behandelt den Transfer des Programms UEBUNG 1 von der MCZ-Diskette in den DM-RAM. Der Empfang in der SIO/2 des DM-Boards ist mit der Überprüfung einer Checksumme verbunden.

Der Z 8000 DM-Monitor enthält überdies 13 Befehle, die das Ändern von Speicher- und Registerinhalte, das Einbinden von E/A-Routinen (PIO und CTC) und das Setzen von Software-Breaks gestatten. Folgende Liste diskreter Monitorbefehle verschafft einen Überblick:

- BREAK <Adresse> [<n>]

  z.B. BREAK 5100 1

  setzt einen definierten Anhaltepunkt der Programmausführung an der Stelle % 5100. Mit 1 wird angegeben, daß es sich um den ersten BREAK handelt. Die Adresse muß als geradzähliger Wert angegeben werden.

- NEXT [<n>] z.B. NEXT 1

  ermöglicht die Ausführung eines Programms in Einzelschritte oder in der mit n angegebenen Schrittweite.

- GO , G

  ermöglicht den Start bzw. die Fortsetzung des Programms ab dem Wert, der im Programmzähler der CPU momentan steht.

- JUMP <Adresse>
  z.B. JUMP 5100
  setzt durch einen Sprung den Programmzähler der CPU auf den angegebenen Wert. Vorhergehende Registerwerte und der FCW werden auf den Stapel gelegt.
- FILL <Adresse 1><Adresse 2>
  <Datenwort>
  z.B. FILL 5010 5020 1ABF
  bewirkt das Laden des Speicherwerks von der Adresse 1 bis zur Adresse 2 mit dem angegebnen Datenwort, hier % 1ABF.

Ferner die Befehle: I/O PORT, MOVE, COMPARE, PUNCH TAPE, READ TYPE und zuletzt QUIT. Der letzte Befehl "QUIT" stellt einen Übergang vom Monitor- in den Transparent-Modus dar, d.h. es wird anschließend wieder mit dem MCZ-/ZDS-System bzw. dem RIO gearbeitet.

Das Ziel der Kombination eines Z 8000 DM mit einem MCZ-/ZDS-System ist die Gegebenheit einer leichten Eingabe und Entwicklung von Z 8000-Assemblerprogrammen und entsprechend ablauffähigen Z 8000-Prozeduren. Eine Prozedur ist das Aneinanderreihen von Z 8000-Maschinencodes in HEX-Werten, die für die CPU ausführbar sind. Die syntaktische Prüfung ist dabei der erste Schritt. Man nennt diesen Vorgang "assemblieren". Man beginnt die Eingabe des gewünschten Programms über die Tastatur, indem mit "Q" der Transparent-Modus und "CR" der Eintritt in das RIO ausgeführt wird. Sogleich ruft man den Text-Editor auf (% EDIT < Programmname >). Der Programmname gilt als grundsätzliches und künftiges Kennzeichen eines Programm-Moduls, das folgend als ASCII-File in die Disketten-Dokumentation aufgenommen wird (wenn es als fertiges auf die Diskette übernommen wurde).

Ist ein Assembler-Programm (Beispiel in Kapitel 6) eingetippt worden, wird mit

Z 8000ASM <Programmname>

das Programm assembliert, d.h. auf die syntaktischen Fehler überprüft. Dieses sehr wichtige Dienstprogramm ist im Rahmen der Z 8000 Software-Entwicklungshilfen über das RIO-Betriebssystem anzuwenden. Nach dem Programmdurchlauf werden die Fehler im eingetippten Assembler-Programm ausgegeben. Die Korrektur wird durch den Ausdruck der fehlerhaften Zeile und anschließend mit CHANGE

> C / altes Zeichen / neues Zeichen /

ausgeführt. Nach erfolgten Korrekturen wird das Programm nochmals assembliert. Ist daraufhin das Programm ohne syntaktischen Fehler ausgewiesen worden, hat man ein ablauffähiges ASCII-File. Der Z 8000-Assembler (Crossassembler) ist nach der höheren Programmiersprache PLZ orientiert. Er enthält Elemente wie DO, IF, THEN, ELSE.

Als zweites wichtiges Hilfsmittel einer Programmerstellung ist der Z 8000 Linker zu nennen. Er hat die Aufgabe, ein komplettes, größeres Programm, bestehend aus mehreren Programm-Moduln, zu einem Objekt-Modul zusammenzubinden. Zusammenbinden bedeutet, daß für einen betreffenden Fall alle Programmcode- und Datenelemente getrennt aneinandergereiht werden, die durch die einzelnen Programm-Moduln gegeben sind. Es werden insgesamt zwei Speicherabschnitte (Code und Daten) der beteiligten Programm-Moduln gebildet. Der Vorteil besteht darin, daß beliebige Einzelroutinen, die irgendwann einmal geschrieben wurden, in irgendeinem größeren Anwenderproblem leicht verwendet werden können.

Zuletzt besteht die dritte Phase der Programmerstellung darin, aus dem Objekt-Modul eine für die Z 8000-CPU ablauffähige Prozedur zu erstellen. Diese Aufgabe wird mit dem Programm "Z 8000-Imager" erreicht. Dieses Programm erzeugt das Procedure-File in Form von Z 8000-Maschinencodes. Es sollte hier nur noch einmal in Erinnerung gebracht werden, daß die drei wesentlichsten Software-Entwicklungshilfen: PLZ As-

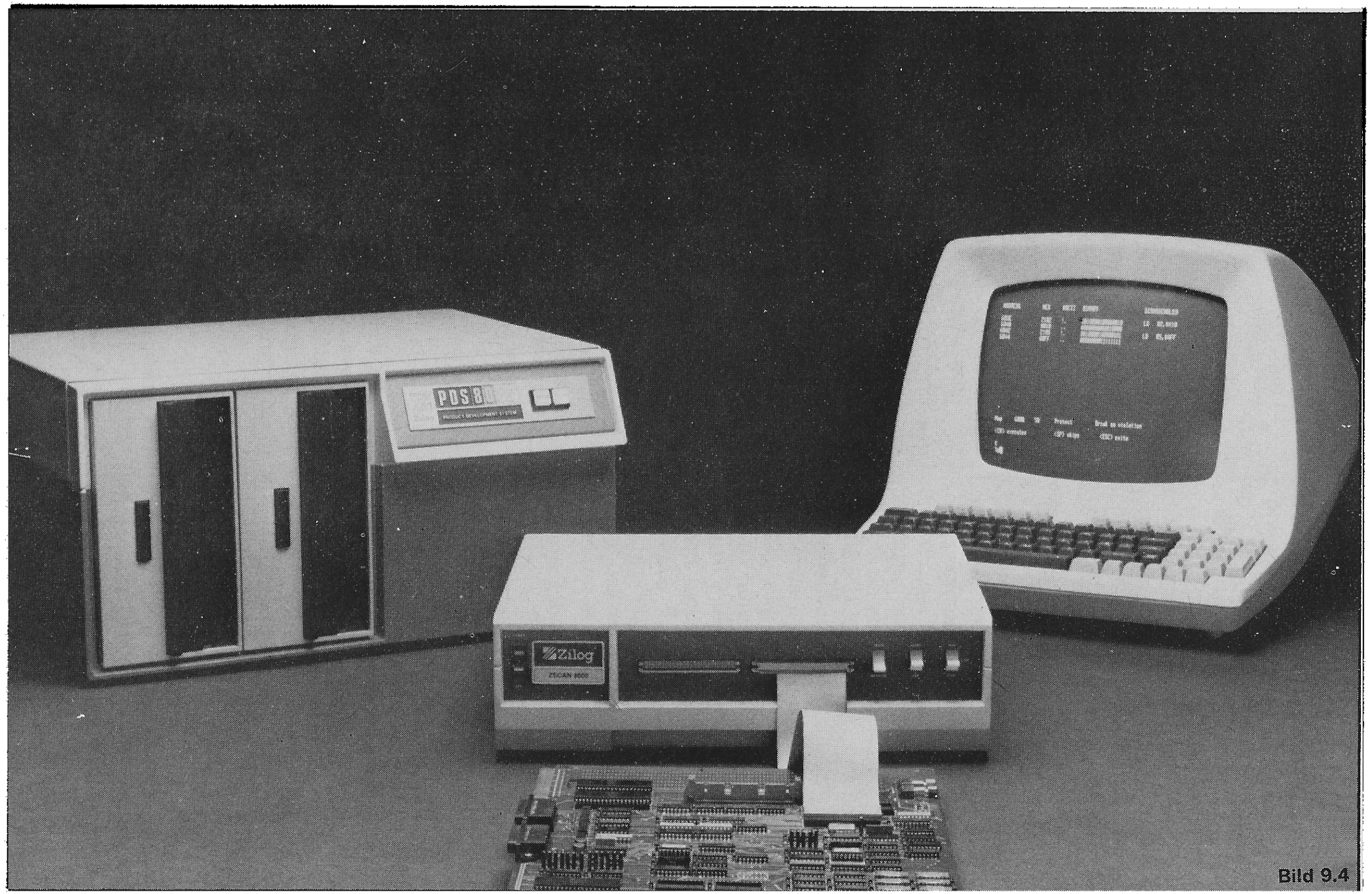


Bild 9.4

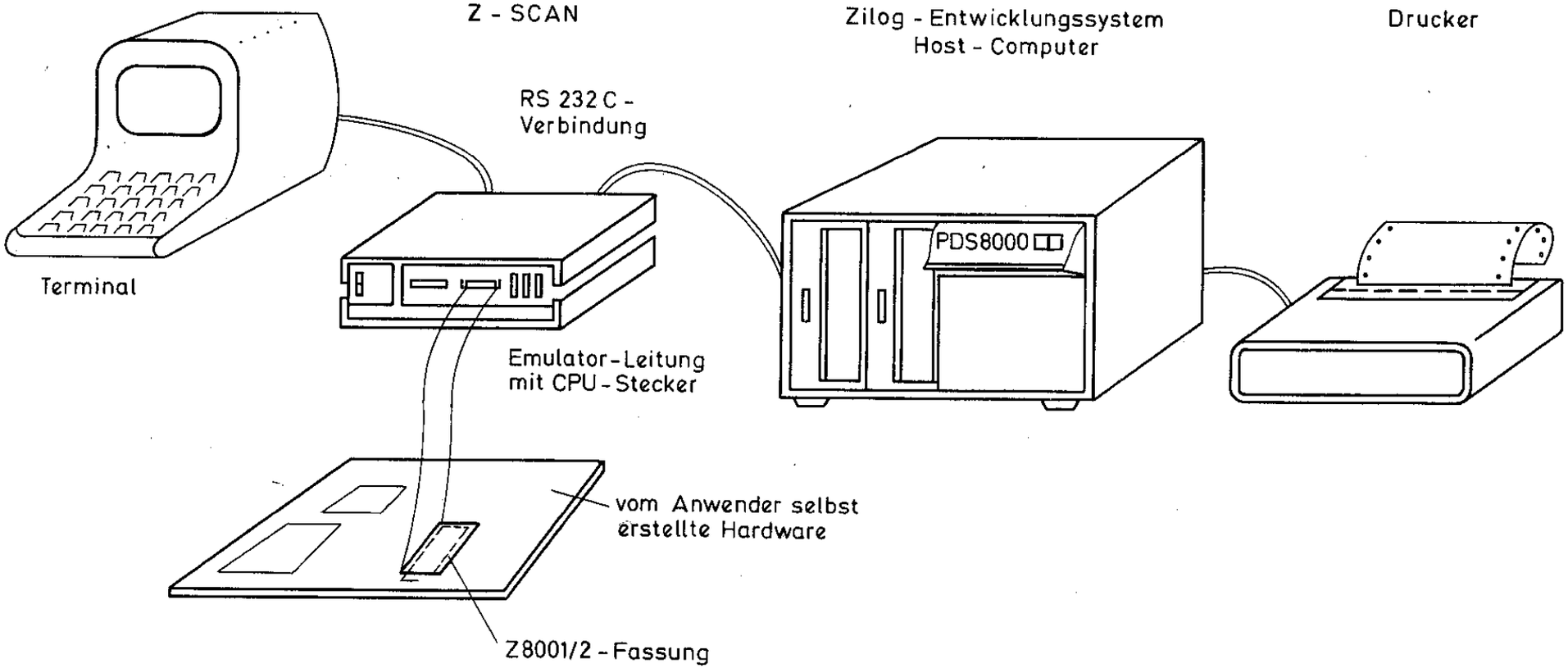

Z - SCAN
Zilog - Entwicklungssystem
Host - Computer
Drucker
RS 232 C -
Verbindung
Terminal
PDS8000
Emulator-Leitung
mit CPU-Stecker
vom Anwender selbst
erstellte Hardware
Z8001/2 - Fassung

Bild 9.5

sembler, Linker und Imager, alle im Z 80-Source-Code geschrieben sind. Diese Programme laufen alle auf den Zilog-Entwicklungssystemen. Damit sollte nochmals unterstrichen werden, daß man hier keinesfalls mit unverhältnismäßigem Aufwand arbeitet.

Hat man nun endliche das Z 8000-Procedure-File, wird dieses Programm mit LOAD, wie schon erwähnt, in das Z 8000-DM transferiert. Der PC und das PCW der CPU wird gesetzt (Anfangsadresse % 5000 und FCW, z.B. % 4000) und sodann kann das Programm gestartet werden, das sich im dyn RAM des DM-Boards befindet. Damit ist die Erstellungsphase eines Z 8000-Programms mit den angebotenen Zilog-Hilfsmitteln kurz geschildert.

Wie geht es dann weiter?
Diese Frage ist sicher nicht unberechtigt. Das Z 8000-DM dient als Testbasis für Software, die einmal in einer selbstgebauten Hardware laufen soll. Um das entwickelte Programm in einer selbtgebauten Hardware testen zu können, muß eine residente Form des Programms erstellt werden. Ein Testprogramm für ein selbstentworfenes Z 8000-System wird in zwei EPROMs implementiert. Zwei Bausteine sind erforderlich, weil das Programm in der Wortbreite von 16 bit vorgegeben ist. 16 bit EPROMs gibt es bisher nicht. Mit diesem Programm in Festwertspeichern kann ein beliebiges Z 8000-Hardwaresystem getestet werden.
Diese Methode gilt, trotzdem sie häufig angewandt wird, als ziemlich umständlich. Die "off-line"-Form der Programmentwicklung über die "Krücke" EPROM verlangsamt die Erstellungsphase doch beträchtlich. Der Sinn nach einer effektiven "on-line"-Entwicklungsmethode stand deshalb den Herstellern schon sehr früh nahe. Das Stichwort hierfür lautet:

ECHTZEIT-EMULATION.

Echtzeit-Emulation bedeutet, daß die selbstentwickelte Hardware direkt mit der Software aus dem Entwicklungssystem versorgt wird und in dem Ablaufzyklus "gefahren" werden kann, der der normalen Arbeitsgeschwindigkeit entspricht. Normal bedeutet: Die Anwender-Hardware wird mit 250 ns bzw. 166 2/3 ns Zykluszeit simuliert, jedoch nicht mit der CPU, sondern dem Entwicklungssystem als CPU. Diese sogenannten ICE-Systeme (In-Circuit Emulation) gibt es bereits für viele 8 bit Hardwaresysteme. Die Firma Zilog bietet ein System an, das jetzt auch für den Z 8000 die Simulation einer Z 8000-Anwenderhardware gestattet. Das Gerät hat den Namen ZSCAN 8000. **Bild 9.4** zeigt das Gerät und zugleich ein wenig vom Prinzip. Eine 40polige Flachbandleitung mit einem CPU-Stecker (bzw. 48polig für die Z 8001-CPU) wird von dem ZSCAN in die CPU-Fassung des Anwendersystems gesteckt. Damit ist die genannte "on-line"-Verbindung hergestellt. Die Programmentwicklung spielt sich in gleicher Art ab, wie sie vorher geschildert wurde. Man gehe davon aus, daß ein PDS 8000-Hostsystem angeschlossen ist, wie es in **Bild 9.5** dargestellt ist.

1. Eingabe des Programms mit Text-Editor
2. Assemblierung mit Z 8000ASM
3. Eventuelle Korrektur
4. Eventuell Linken, wenn mehr als ein Programm-Modul in der Test-Prozedur verwendet werden
5. Erzeugung des Procedure-Files
6. Laden des Programms in das ZSCAN

Der Unterschied zum Z 8000 DM-Board besteht für die weitere Bearbeitung darin, daß ein Procedure-File, das sich im RAM des ZSCAN befindet, unmittelbar in der Anwenderhardware zur Ausführung kommt, weil es sich auch im RAM des Anwendersystems befindet. Speicher-CPU-Operationen können in 100 % realer Zeit simuliert werden. Test-Breaks lassen sich durch einen Hardware-Trigger in acht verschiedenen Kombinationen setzen. Damit lassen sich mit Hilfe von Logik-Analysatoren Signalzustände in bestimmten Phasen erfassen (bei besonders kritischer Anwenderhardware).

9.2 EVALUATION-BOARD AM 96/4016 UND RTE 16/8050 VON AMD VON AMD

Auch die Firma AMD bemüht sich in gleicher Weise, für den Z 8000 Systeme anzubieten. Mit dem Evaluation-Board AMC 96/4016 wird ein praktisches Übungsgerät angeboten. In Verbindung mit dem AMC-Hostcomputer AM SYS 8/8 wird dieses kleine Z 8000-System zu einem hochwertigen Entwicklungssystem.

Das Evaluation-Board 96/4016 ist ein autonomes System auf einer Platine nach einer amerikanischen Norm (12,0" x 6,75", Zoll, = 305 x 172 mm²) und dem ebenfalls normierten MULTIBUS. **Bild 9.6** zeigt eine Fotografie dieser Platine, die mit sechs Steckverbinderleisten an den Platinenkanten ausgestattet ist. Auch bei diesem System kommt die nicht segmentierte CPU Z 8002 zur Anwendung. Die Gliederung des Systems ist in dem Bus-Interface für Adressen und Daten, einem Festwert-Speicherbereich, einem dynamischen Arbeitsspeicherfeld, 24 parallelen E/A-Leitungen, zwei RS 232 C-Serienschnittstellen (plus einer 20 mA-Stromschleife für TTY) und letztlich einem Tastatur-Interface gegeben. Dieses Board ist in der Weise interessant, weil es mit einer Tastaturplatine mit 56 Tasten und 20 Zeichen alphanumerischer LED-Anzeige (als Option fertig vorbereitet) ohne weitere Systemunterstützung zu betreiben ist. Die Verbindung der CPU-Platine mit der Tastatur erfolgt über eine 26polige Flachbandleitung. **Bild 9.7** zeigt diese Verbindung. Diese Systemkonfiguration stellt bisher die einfachste Art dar, mit einem Z 8000-System Programmiererfahrung zu sammeln.

Das Blockdiagramm in **Bild 9.8** zeigt die Hardware des Evaluation-Boards etwas genauer. Das Bus-Interface zur Trennung von Adressen und Daten wird mit dem Baustein 8173 und 8104 realisiert, wie es in Bild 4.3 genau gezeigt wurde. Der Festwert-Speicherbereich ist mit sechs Steckplätzen ausgeführt. Zwei EPROMs (2716) enthalten das Monitorprogramm des Boards. Der Arbeitsspeicher ist mit dynamischen RAMs des Typs Am 9050 belegt (4 K x 1 bit). 16 Bausteine auf dem Board ergeben 4 K Worte. Die 24 parallelen E/A-Leitungen ergeben sich mit Hilfe des Interface-Bausteins 8255A, der mit seinen Busanschlüssen D0 . . . D7 an der unteren Hälfte des CPU-Datenbusses anliegt (siehe auch Bild 4.60). Die serielle Kommunikation wird mit zwei Bausteinen des Typs 9551 (8251) hardwaremäßig ausgeführt. Dieser Baustein wandelt parallele Daten (Bytes) in ein serielles, asynchrones Datenformat (mit Start- und Stopbit), das in relativ weiten Grenzen programmierbar ist. Beide Bausteine sind zum Betrieb zweier getrennter, asynchroner E/A-Schnittstellen des Typs RS 232 C ausgebaut, wobei eine noch mit einer 20 mA-Stromschleife versehen ist. Mit dem Baustein 8253 steht noch ein programmierbarer Zeitgeber zur Verfügung, der drei unabhängige 16-bit-Zähler beinhaltet. Man kann diese im Binär- oder BCD-Modus abwärts zählen lassen (mit max. 2 MHz). Zwei Zähler werden zur Baudraten-Steuerung der seriellen Schnittstelle benutzt; der dritte Zähler ist für den Anwender frei.

Für die letztgenannten Bausteine 9551 und 8253 gilt, daß alle an das untere Byte des CPU-Datenbus angebunden sind.

Die Breakpoint-Logik besteht aus zwei 8-bit-Registern mit CLR und zwei 8-bit-Komparatoren. Damit läßt sich ein Breakpoint setzen (Breakpoint-E/A-Adresse % 0FD4). Dieser Hardware-Breakpoint wird gesetzt, indem über den Befehl OUT an der E/A-Adresse wiederum eine 16-bit-Adresse über den Datenbus ausgegeben wird. Damit werden die Register geladen und liefern damit den Vergleichswert zu den laufenden Programmadressen. Bei Koinzidenz

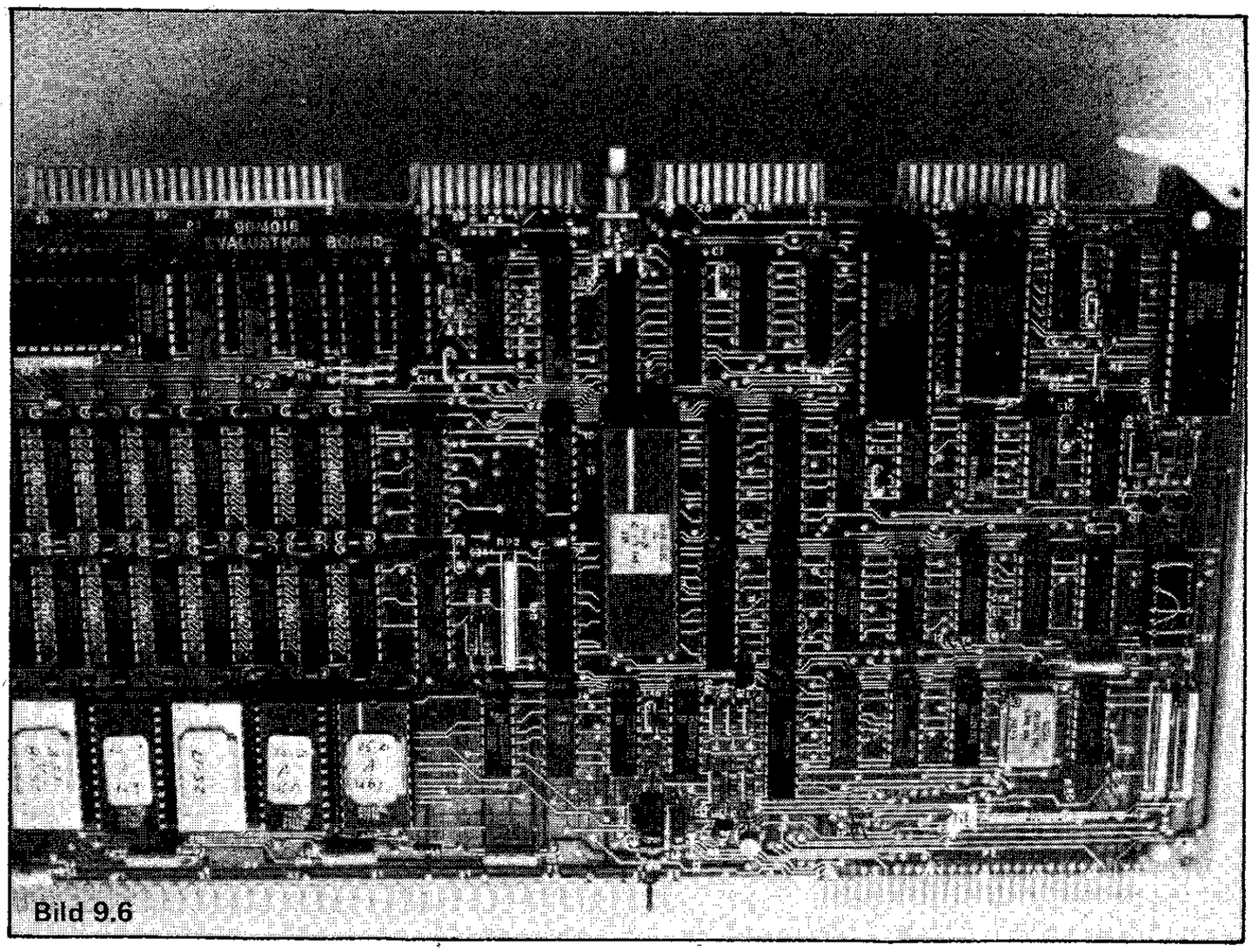


Bild 9.6

wird aus der Break-Logik ein $\overline{\text{NMI}}$ an die CPU gesendet. Für den Bediener des Boards ist die Initialisierung der Break-Adresse mit dem Befehl E (%) nnnn zu erledigen. E steht für Hard-Break setzen und nnnn für die gewünschte Adresse (HEX-Code).

Damit ist der Monitor des AMC-Boards angesprochen. Die Eingabe B ermöglicht auch das Setzen von Software Break-points.

Die Funktionen des Monitors beschränken sich auf die bekannten Editier-Funktionen (Eingabe in Klammer):

Anzeige:
Speicher (D x, y); Speicher plus Änderung (S x); Programmzähler (XP); alle CPU-Register (X); Wortregister (X n); Flags (XF)

Schreiboperationen:
Speicher (F x, y, z); Programmzähler (P=x); Register (RHn=x) (RLn=x) (Rn=x)

Initialisierung:
Flags setzen (C n) (D n) (H n) (V n) (P n) (S n) (Z n)

[x, y, z bedeuten HEX-Werte,
n → Dezimalwert]

Ferner ist ein Datentransfer-Modus im Monitor inbegriffen, der eine Programmübertragung von einem SYS 8/8-Laufwerk in das Board und zurück gestattet.

LOAD d: < Programmname >, nnnn

transferiert von Disk-Laufwerk Nr. "d" das Programm < Name > in den RAM des Boards ab der Adresse nnnn (HEX).

SAVE d:
< Programmname >, nnnn, mmmm

transferiert vom RAM des Boards zwischen Adresse nnnn und mmmm das Programm < Name > zum Laufwerk Nr. "d" des SYS 8/8.

Der Arbeitsspeicher des Boards ist zwischen den Adressen **% 4000** und **% 6000** eingegliedert. Davon werden die ersten

256 Byte als scratchpad (Stapelspeicher) verwendet.

Neben einzelnen Ausführungsanweisungen enthält der Monitor auch einen Aufruf für einen Assembler, mit dem das Board zusätzlich bestückt werden kann. Mit

ASM 4100

kann man unmittelbar Mnemonic-Zeichen in den Maschinencode umsetzen lassen. % 4100 zeigt auf die Anfangsadresse im Arbeitsspeicher, an der beginnend der Maschinencode abgespeichert wird, aus dessen Befehlseingabe in Assemblersprache er erzeugt wurde. Neben Mnemonic-Zeichen übersetzt der Assembler symbolische Labels, symbolische oder absolute Operanden (HEX) in einer Einwegausführung und legt die Maschinencodes mit steigender Adresse im Arbeitsspeicher an. Der Befehl: Subtrahiere vom Inhalt in R 0 den Inhalt der Speicherstelle, dessen Adresse in R 10 steht, —— lautet nach dem Assembler

SUB R 0, R 10 ;

Der Strichpunkt am Schluß bezeichnet das Ende des Statements. Danach wird (CR) gedrückt. Die Form

SUB R 0, @ R 10

ist aus Abschnitt 3.3 bekannt. Für die indirekte Adressierung beim AMC-Assembler steht statt @ R 10 die Form R 10 . Daran muß man sich eben ge-

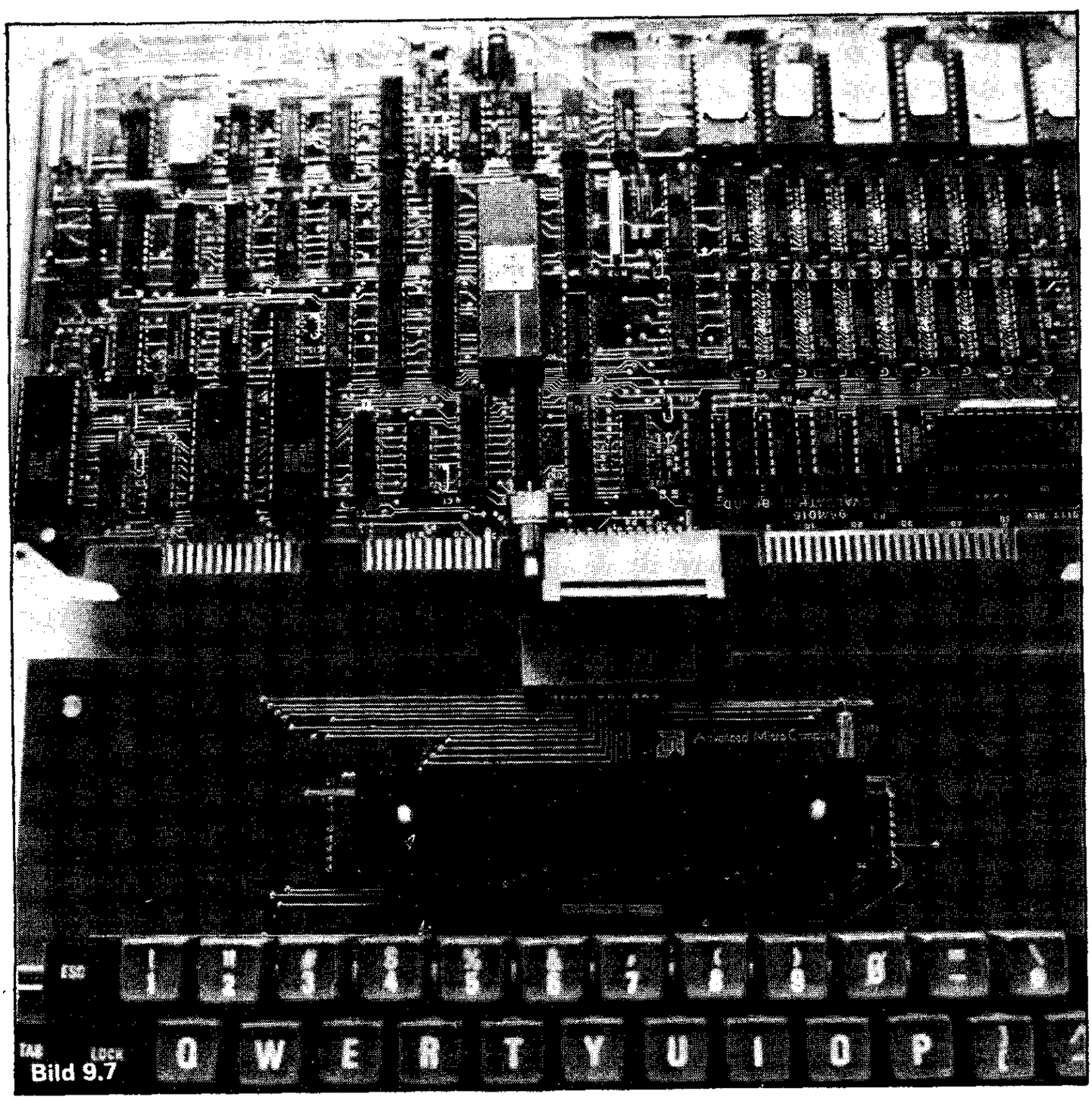


Bild 9.7

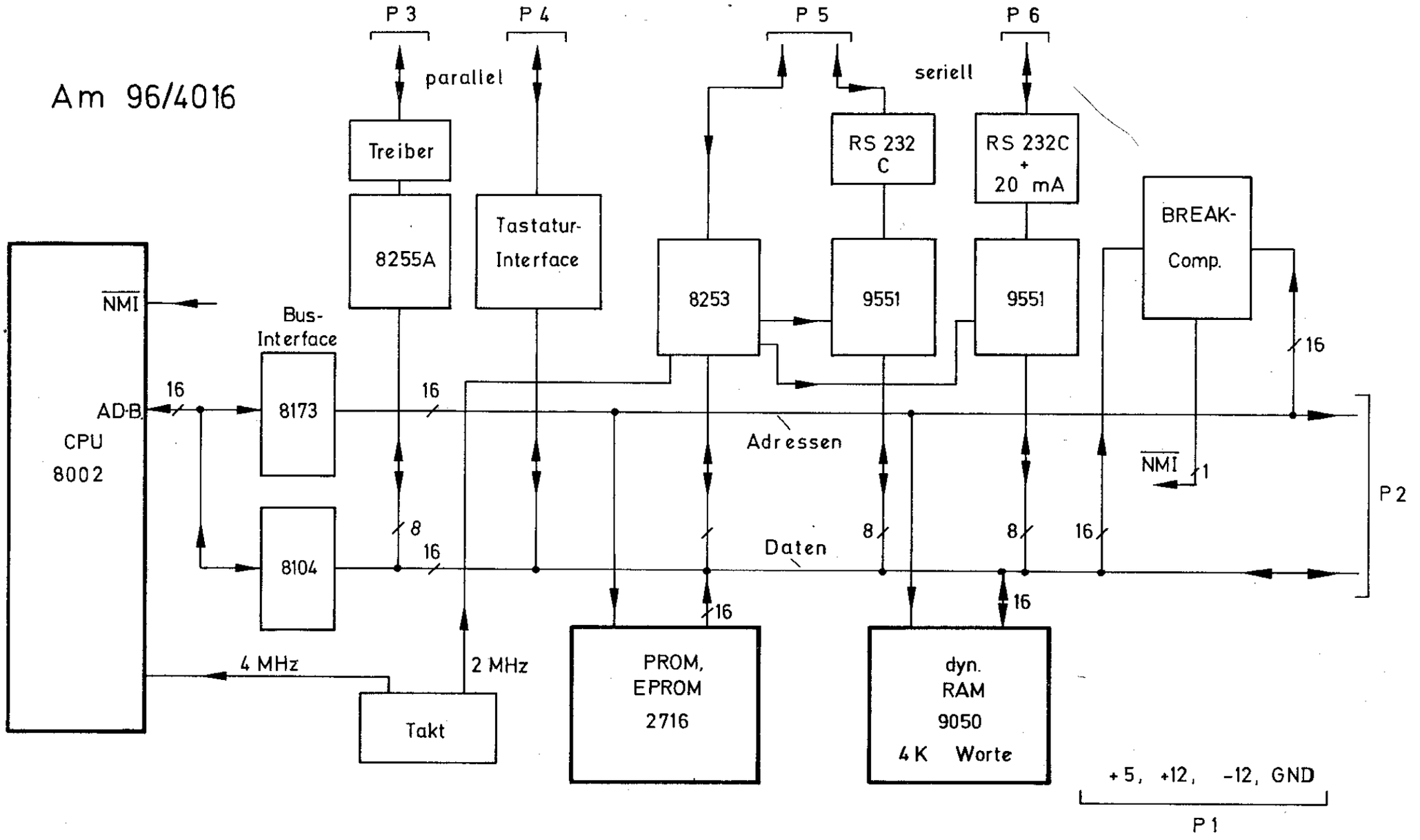

Am 96/4016
P 3
P 4
P 5
P 6
parallel
seriell
Treiber
8255A
Tastatur-
Interface
RS 232
C
RS 232C
+
20 mA
BREAK-
Comp.
8253
9551
9551
NMI
Bus-
Interface
16
AD-B
CPU
8002
8173
16
16
Adressen
NMI
1
P 2
8
8104
16
Daten
8
8
16
16
16
4 MHz
2 MHz
Takt
PROM,
EPROM
2716
dyn.
RAM
9050
4 K Worte
+5, +12, -12, GND
P 1

Bild 9.8

wöhnen. Für den AMC-Assembler gelten noch einige andere Vereinbarungen, die in Abschnitt 3.3 anders definiert sind. Der AMC-Assembler ist gegenüber dem Zilog-Disk-Assembler resident und zeilenorientiert. Er ist in vier EPROMs des Typs 2716 untergebracht und ist folglich 8 KB lang. Es wurde für den Z 8000 im Z 8000-Quellcode entwickelt. Somit enthält das Am 96/4016-Board 12 KB Festwertspeicher bei vollem Ausbau.

Die Firma AMC (Advanced Micro Computers) hatte, ebenfalls wie die Firma Zilog, frühzeitig ein Produkt für die Hard- und Softwareentwicklung mit beiden Z 8000-Typen im Auge. Dieses allgemein mit "Support Package" und mit der spezifischen Bezeichnung RTE 16/8050 versehene System ist sowohl ein ICE (In-Circuit Emulator) wie auch ein Logik-Analysator. Das Emulator-System erlaubt eine 100 %ige Zeit-Simulation von Anwenderhardware beider CPU-Typen, wie auch einen Einzelschritt-Test. Die Besonderheit dieses Entwicklungssystems ist, zugleich als Logikanalysator für die zu testende Hardware mitzuwirken. Ein Adapter mit 8 Testleitungen ermöglicht die Aufnahme von 8 Signalen zur Kontrolle kritischer Signalzustände und Laufzeiten während eines Maschinen- oder Befehlszyklus im Anwendersystem. Die acht Testleitungen dienen als Triggerpunkte. Die Busaktivitäten während der Testphase werden in 256 Worten à 46 bit festgehalten (23 Adreßleitungen, 16 Datenleitungen, 7 Statusleitungen). Acht schnelle Komparatoren werden durch die externe Triggerung ausgelöst, zu dessen jeweiligen Zeitpunkten wichtige Buszyklen festgehalten werden. Man kann beispielsweise eine bestimmte Bit-Permutation von Daten bei einer vorgegebenen Adresse testen, ferner den Ablauf von E/A-Zyklen, eine Speicher-response mit und ohne Wartezyklus und vieles mehr. All diese Ereignisse kann man dann in Ruhe auf dem Bildschirm des Terminals abrufen, analysieren und feststellen, welche (eventuellen) Fehler man in der Eigenbau-Hardware begangen hat. Es ist wohl selbstverständlich, daß man mit Hilfe eines solchen Systems individuelle Z 8000-Hardware relativ schnell "lauffähig" machen kann.

Das Z8000-System für die 80er Jahre

10

10.1 SYSTEMBAUSTEINE FÜR ALLGEMEINE UND SPEZIFISCHE ANWENDUNGEN

Die beiden CPU-Bausteine Z 8001 und Z 8002 sind als die zentralen Elemente einer "großen Familie" anzusehen. Für eine einfache Hardwareentwicklung werden zu den CPUs periphere Bausteine angeboten und noch hinzuentwickelt, die **eine** Eigenschaft gemeinsam haben: Sie passen zueinander. Zueinander passen bedeutet, daß sie Signalanschlüsse aufweisen, die in ihrer Bezeichnung einheitlich und folglich in der logischen Funktion identisch sind. Basierend ist die Signalformation, die beide CPUs zu Grunde legen. Über den Zeitmultiplex-Adreß-/Datenbus hinaus sind dies die Steuersignale $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, B/$\overline{W}$ und eine Reihe anderer, die zusammen eine einfache, durchschaubare Systematik ergeben. In **Bild 10.1** sind diese Signale nochmals zusammengefaßt, die den Z 8000-Bus bilden. Alle wesentlichen Systembausteine haben und werden die entsprechend ihrer Funktion notwendigen Bussignale als Anschlußbezeichnungen tragen. Ein weiteres Merkmal der Systembausteine ist, daß der Multiplexbus im jeweiligen Baustein durch $\overline{AS}$ und $\overline{DS}$ von der CPU in Adressen und Daten getrennt wird. Neben der 8010-MMU und dem 8016-DMATC weisen auch die verschiedenen Schnittstellenbausteine und andere diese interne Trennung auf. Aus diesem Grund wird ein Busanschluß sehr einfach. Die beiden wichtigsten Peripheriebausteine sind der Z 8034 UPC für parallele Schnittstellen und der Z 8030 SCC für serielle Schnittstellen. Die Standardschaltung eines Z-Bus-Anschlusses dieser Bausteine zeigt **Bild 10.2.** Busseitig wird generell das untere Byte benutzt. Die durch B/$\overline{W}$ gleich Hi angezeigten Byte-Daten werden immer mit einer ungeraden Adresse (E/A-Adresse) übertragen. Peripheriebausteine mit einem Z-Bus-Anschluß werden überdies in eine "daisy chain"-Verbindung eingegliedert. Jeder Baustein wird in seinem Vektorregister quasi mit einer Kennmarke versehen. Hat ein Baustein die Anerkennung seiner Anforderung erfahren, sendet der Baustein diese Kennmarke aus und zeigt damit an, daß er eine bestimmte Interrupt-Serviceroutine benötigt. Die gemeinsame Interrupt-Anforderungsleitung wird man deshalb auf den $\overline{VI}$-Eingang der CPU geben. In Kapitel 2 wird die Anerkennungsprozedur nach dem "daisy chain"-Verfahren hinreichend behandelt. Es sei hier noch angefügt, daß der Anschluß von Bausteinen nach Bild 10.2 ausschließlich asynchron erfolgt. Im Gegensatz zur MMU wird hier grundsätzlich ohne CPU-Takt gearbeitet, weil die CPU-Signale alle entscheidenden Schaltzeitpunkte liefern. Geht man davon aus, daß die CPU generell Busmaster ist, muß man diese Rolle zuweilen auch einem im System befindlichen DMA-Baustein, dem Z 8016, zusprechen.

Der DMA-Baustein ist ebenso in der Lage, die Signale $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, B/$\overline{W}$ und alle weiteren zu erzeugen. Für den DMA gilt jedoch auch die Bezeichnung Peripheriebaustein, weil er auch in eine Kette gebunden werden kann.

Folgend werden einige (ausschließlich) Peripheriebausteine vorgestellt. In **Bild 10.3** ist das Anschlußbild des seriellen Kommunikationsbausteins Z 8030 SCC dargestellt. Der Baustein beinhaltet zwei

unabhängige, serielle Schnittstellen für Voll-Duplexbetrieb bis zu 1 MBit/s Übertragungsrate. Er arbeitet mit einem **eigenen Baudraten-Generator** auf dem Chip und einer digitalen PLL-Taktrückkopplung. Die Übertragung ist wahlweise asynchron (z.B. RS 232C) mit 5 ... 8 Zeichenbits und 1,1 1/2 oder 2 Stoppbits pro Zeichen oder nach dem Bit-orientierten, synchronen Protokoll in HDLC- bzw. dem IBM kompatiblen SDLC-Format. Je ein Baudrate-Generator für einen Kanal erhält durch einen Schreib-/Leseregister eine Zeitvorgabe eingeschrieben, die über die PLL-Schleife konstant gehalten wird.

Bild 10.4 zeigt das Anschlußbild des universellen Parallel-Schnittstellenbausteins Z 8034 UPC. Er verfügt über 20 E/A-Leitungen, von denen 16 bidirektional (Port 1 und 2) und 4 als Einzelbit E- oder A-Leitungen programmiert werden können (Port 3). Der UPC ist als Slave-Prozessor ausgelegt, hat ein 2 KB Programm-ROM und 256 Byte scratchpad auf dem Chip. Der Slave-Prozessor erlaubt die Zwischenspeicherung und Manipulation der über die Schnittstellen

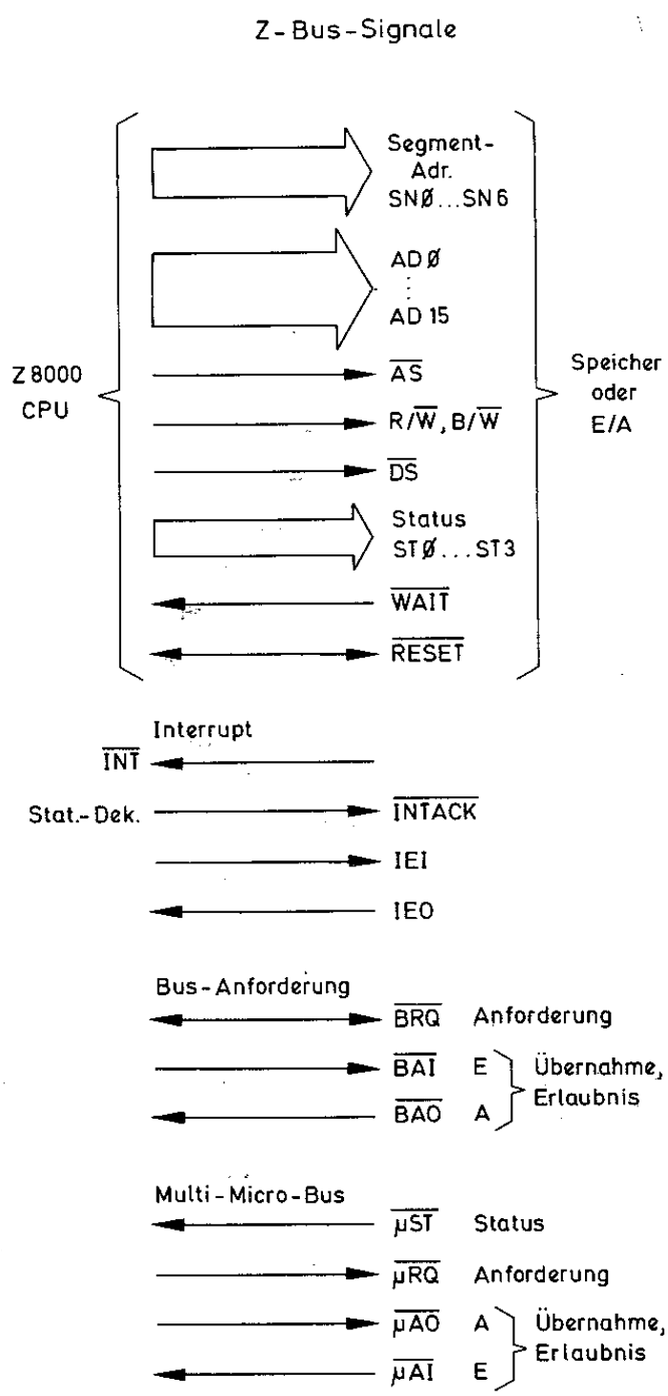
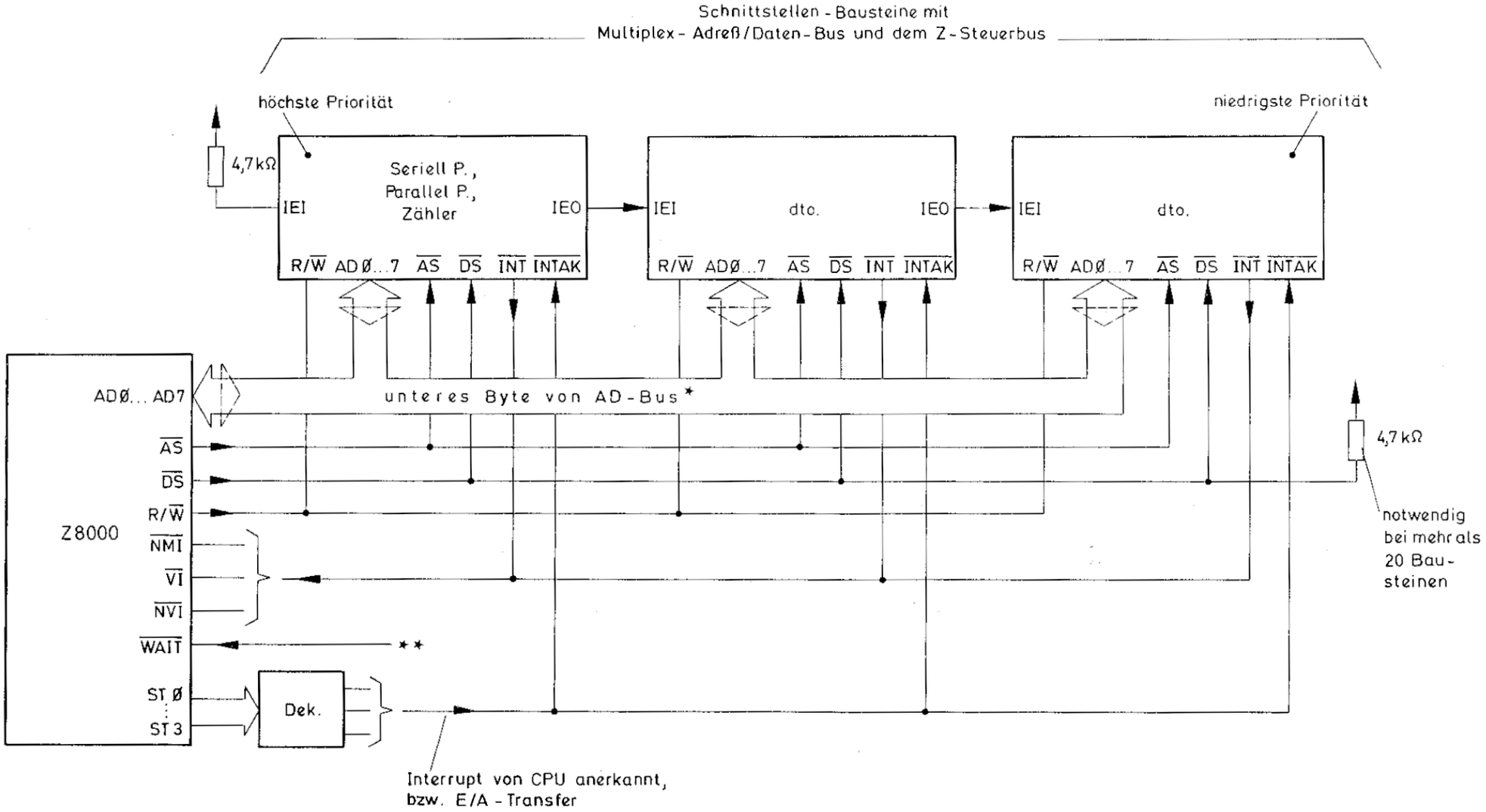


Bild 10.1



* E/A - Transfer plus Initialisierung

** WAIT direkt von Z8034 UPC oder zusätzlicher Logik

Bild 10.2

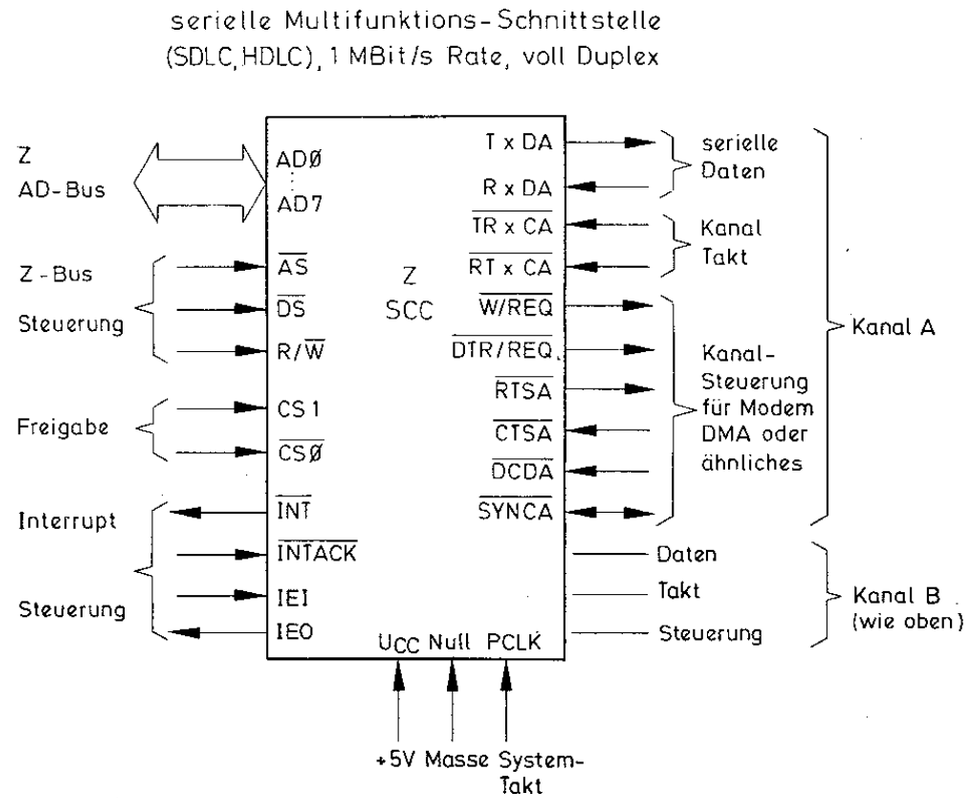


Bild 10.3

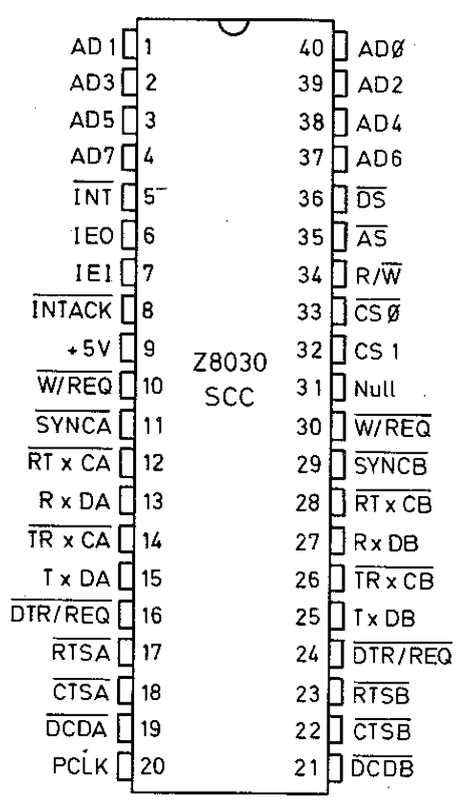


Bild 10.3

übernommenen Daten. Zwei 8-bit-Zähler mit Vorteiler helfen bei der Programmierung von Verzögerungen.

Bild 10.5 zeigt das Anschlußbild des universellen Zähler/Zeitgeber und E/A-Bausteins Z 8036 CIO. Die beiden 8-bit-Ports sind mit doppelten Treiberausgängen versehen und bidirektional. Der spezielle 4-bit-Port (bidirektional) kann entweder mit Tri-State- oder als offener Kollektorausgang programmiert werden. Die drei unabhängigen 16-bit-Abwärtszähler bieten individuelle Programmierbarkeit. Ein Startregister (2 getrennte Bytes) enthält den Wert, der nach jedem Null-Dekrement eines Zählers wieder neu geladen wird. Ein Zählerausgangsregister enthält den laufenden Zählerwert. Jeder Zähler ist über drei Leitungen steuerbar (Takteingang, Gate-Eingang, Triggereingang) und mit einem Ausgang versehen. Letzterer kann als Pulsausgang, Flanken- oder Rechteckausgang programmiert werden. Die vier Zählerleitungen sind für Z 1 an PB4 . . . PB7, für Z 2 an PB0 . . . PB3 und für Z 3 an PC0 . . . PC3 der Portanschlüsse zugänglich.

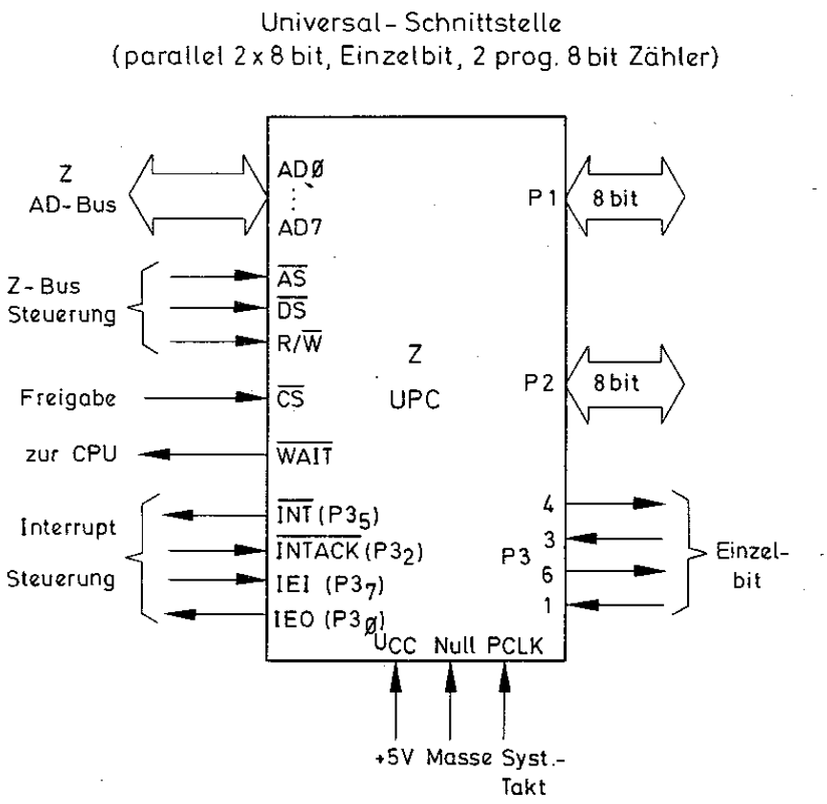


Bild 10.4

Bild 10.6 zeigt das Anschlußbild des Fifo-E/A-Speicherbausteins Z 8038 FIO. Mit diesem Interface-Speicherbaustein der Kapazität 128 Byte werden bidirektionale Datenübertragungen von CPU zu CPU bzw. von CPU zu Peripherie auf rein asynchroner Basis möglich. Dabei spielt eine unterschiedliche Zugriffsgeschwindigkeit des an die jeweilige Interfaceseite angeschlossenen Systems keine Rolle. Der Busanschluß kann sowohl Z-Bus-Format als auch irgendein anderes Format besitzen. Der Datentransfer erfolgt über den Byte-Busanschluß D0 . . . D7 auf Seite 1 bzw. Seite 2. Mit zwei Bausteinen kann auf das 16-bit-Format (256 Worte) ausgebaut werden. Die Transfer-Prozedur wird über einen $\overline{VI}$ oder $\overline{NVI}$ in sieben verschiedenen Arten ausgelöst. Ein "voll" bzw. "leer" des Speichers wird hardwaremäßig angezeigt. Für einen DMA-Betrieb beinhaltet der FIO eine besondere Vorbereitung der internen Logik. Für die Konfiguration der diversen Busformen an Seite A und B sind 12 Kombinationen wählbar, die an den Anschlüssen M0 und M1 für die Seite A

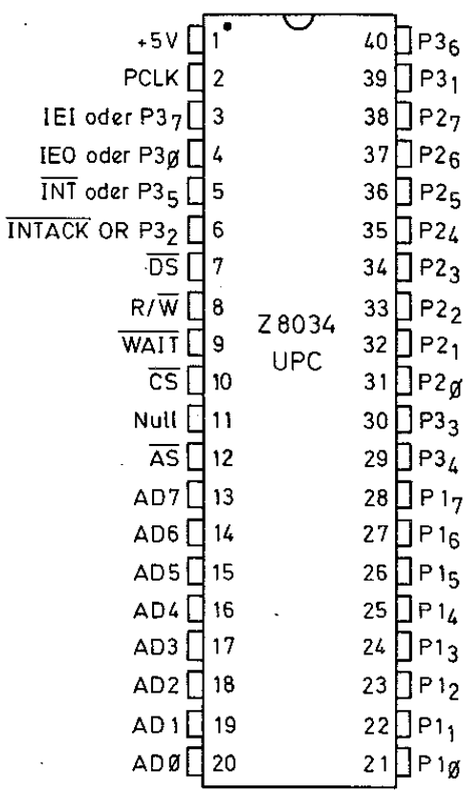


Bild 10.4

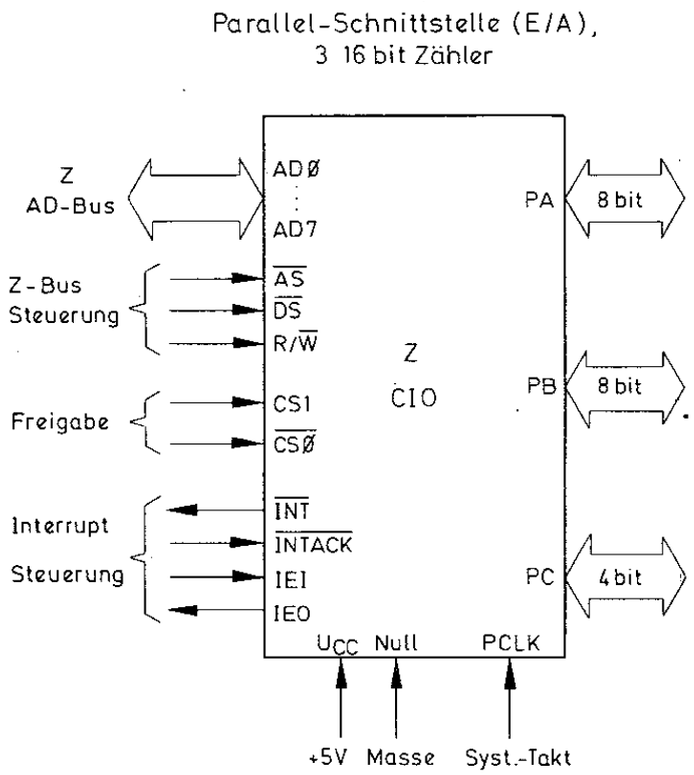


Bild 10.5

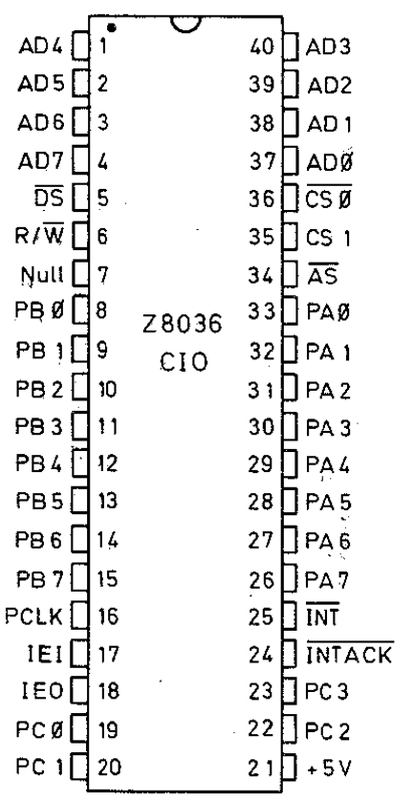


Bild 10.5

und in zwei Bits für die Seite B in einem Steuerregister eingeschrieben werden.

Beispiel:

Seite A:
Z-Bus niederwertiges Byte
Z-Bus höherwertiges Byte

Seite B:
Quittungsbetrieb, 2 Leitungen
Z-Bus höherwertiges Byte

Für den FIO wird auch Speichererweiterungsbaustein angeboten, der einen beträchtlichen Ausbau eines FIO-Interfaces ermöglicht. **Bild 10.7** zeigt die Struktur des Erweiterungsbausteins Z 8060. Dieser ebenfalls 128 Byte umfassende Speicherbaustein nach dem Fifo-Prinzip gestattet ein FIO-Interface praktisch beliebig zu erweitern. **Bild 10.8** zeigt den FIO in einem Interface ohne Erweiterungsbaustein. **Bild 10.9** zeigt die generelle Schaltung eines 512 Byte Fifos.

Für den folgenden Baustein steht die Entwicklung seitens der Firma AMD. Auf dem Bereich der Fehlererkennung wurde der in Kapitel 4 bereits erwähnte

asynchroner bidirektionaler FIFO - Pufferspeicher
(128 x 8 bit)

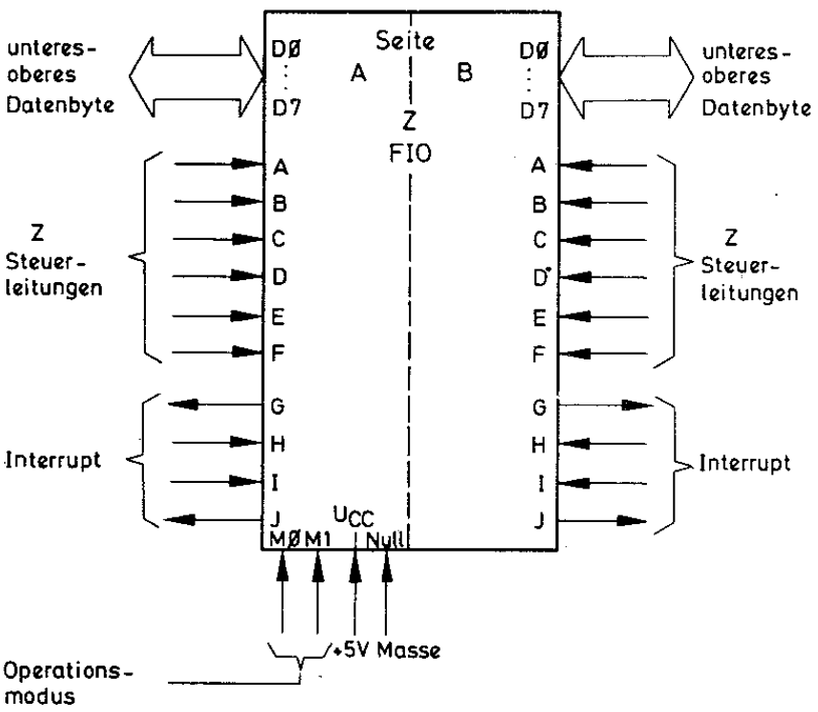


Bild 10.6

8160 EDC vorgestellt. Mit dem Baustein AmZ 8065 BEP (Burst Error Processor) ist ein Baustein für eine generelle Fehlerkorrektur in seriellen und parallelen Datenströmen geschaffen worden. **Bild 10.10** zeigt das Anschlußbild des 8065 BEP. Die Fehlerhäufigkeit von Speicherdaten auf Disketten ist im Steigen begriffen, weil die Datendichten und Übertragungsraten in den letzten Jahren beträchtlich zugenommen haben. Der BEP kann diese Fehler aufspüren und korrigieren. Vier ausgewählte Polynomsätze (IBM 56- und 48 bit, sowie 35- und 32-bit-Polynome) werden nach drei Algorithmen aufgespürt. Die Algorithmen sind algebraischer Natur. An den Eingängen S0, S1 werden die Prüfpolynome für den jeweiligen Datentyp aktiviert, z.B. S0=S1=Hi das Polynom für 48 bit:

$$(x^{13} + 1)\,(x^{35} + x^{23} + x^{8} + x^{2} + 1).$$

So lautet eines der vier Standardpolynome. Die interne Funktion des BEP zeigt (sehr vereinfacht) **Bild 10.11.** Die zu testenden Daten gelangen an D0 . . . D7 in eine Polynom-Teilermatrix und wird nach dem ausgewählten Polynom nach

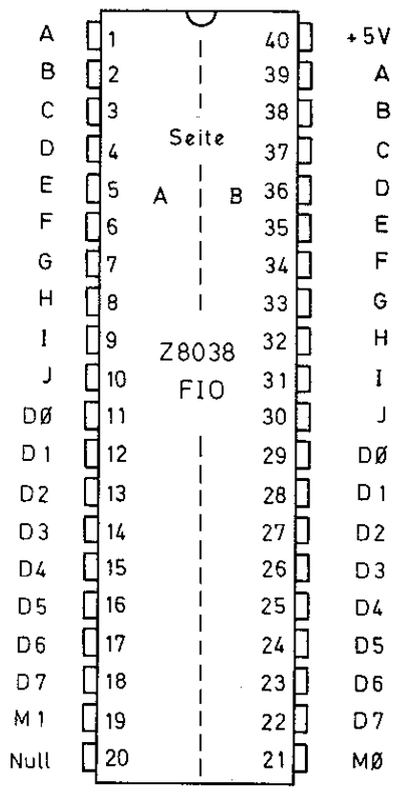


Bild 10.6

FIFO-Speichererweiterung

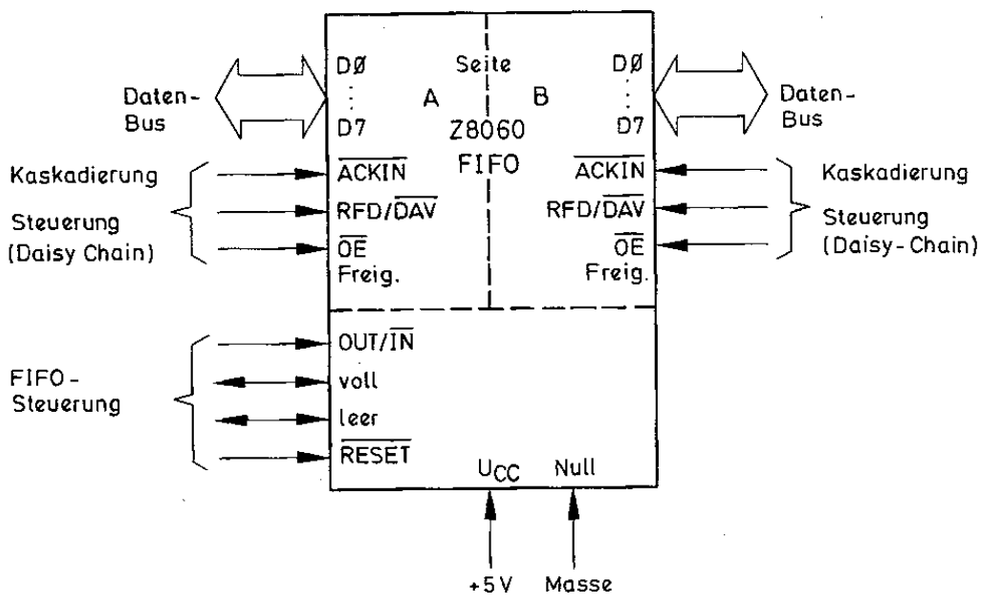


Bild 10.7

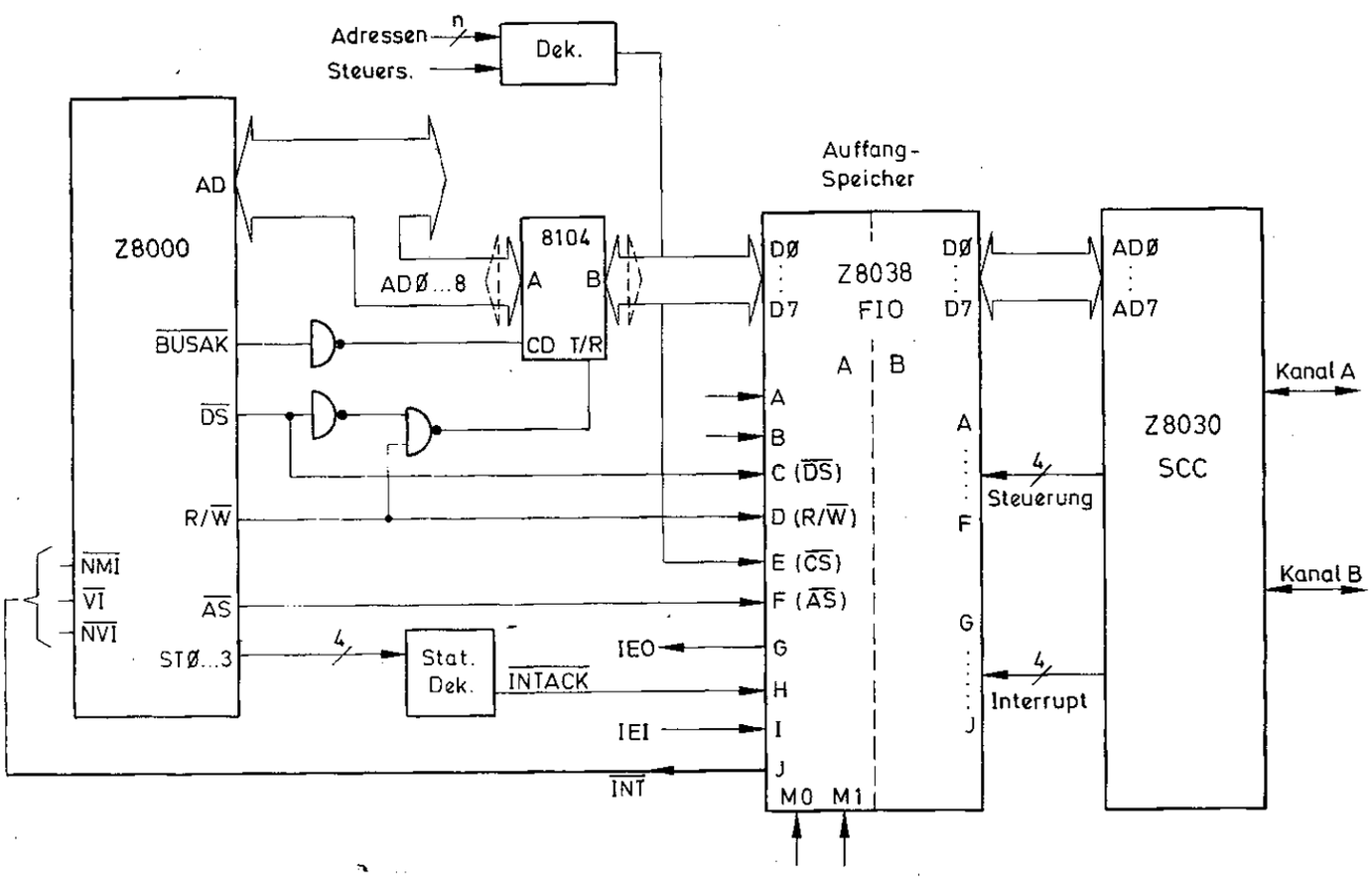


Bild 10.8

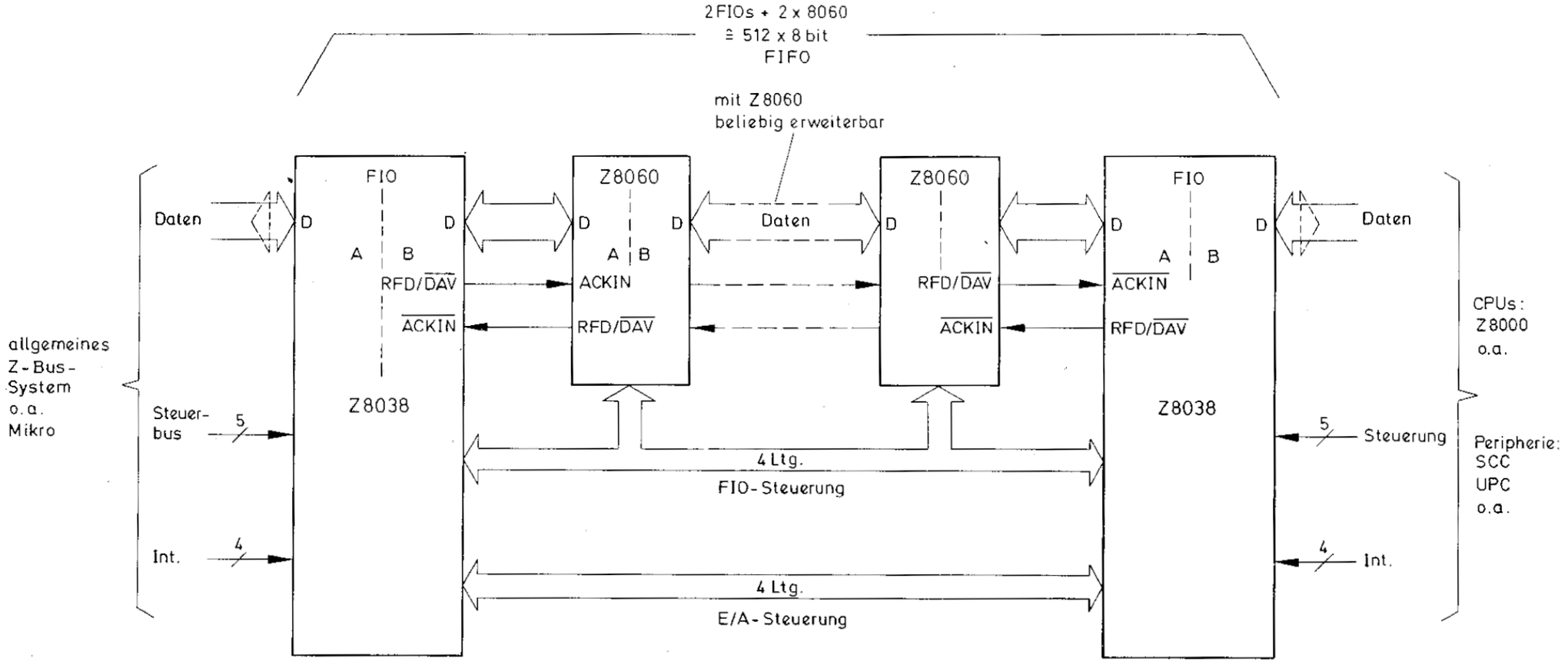

2 FIOs + 2 x 8060
≙ 512 x 8 bit
FIFO
mit Z8060
beliebig erweiterbar
FIO
Z8060
Z8060
FIO
Daten
D
A
B
RFD/DAV
ACKIN
Z8038
Steuer-
bus
5
Int.
4
4 Ltg.
FIO-Steuerung
4 Ltg.
E/A-Steuerung
allgemeines
Z-Bus-
System
o.a.
Mikro
CPUs:
Z8000
o.a.
Peripherie:
SCC
UPC
o.a.
Steuerung

Bild 10.9

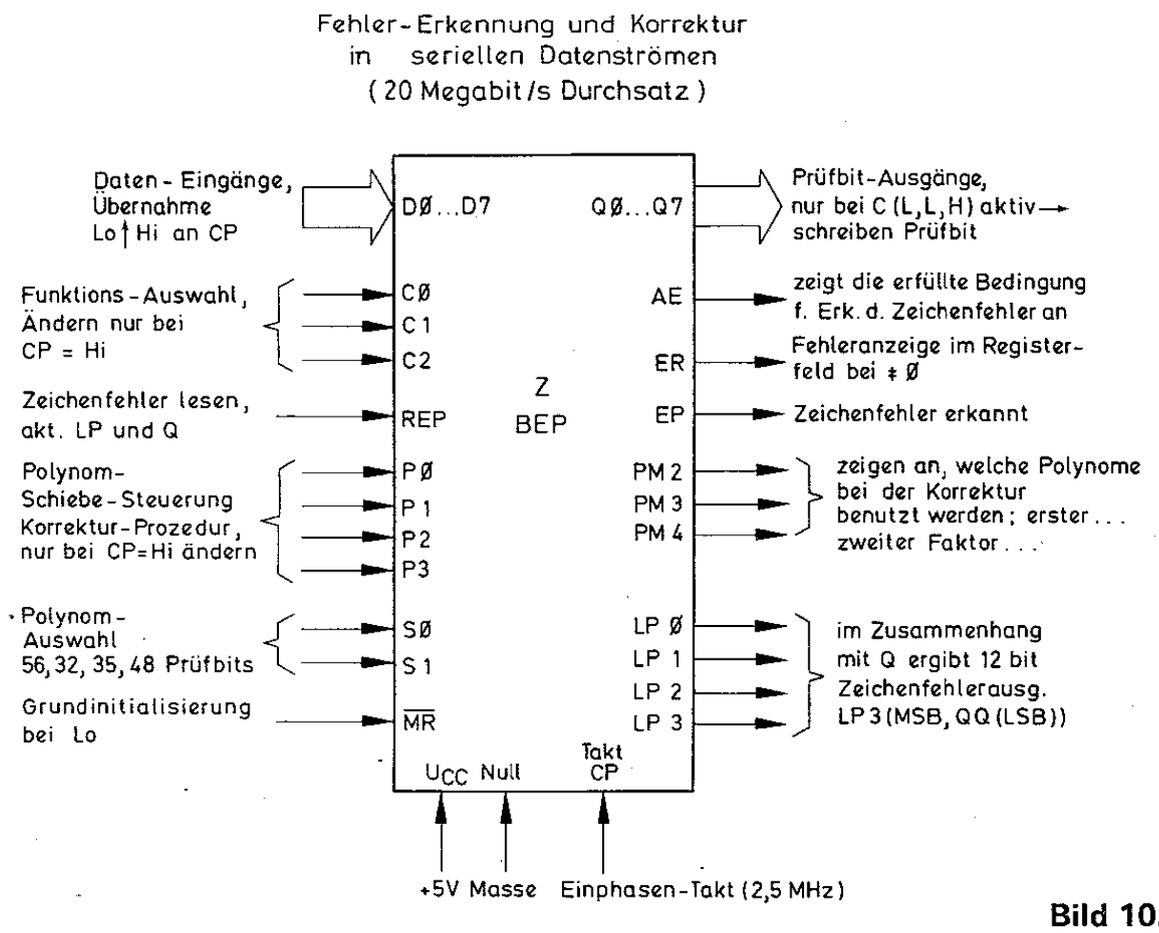


Bild 10.10

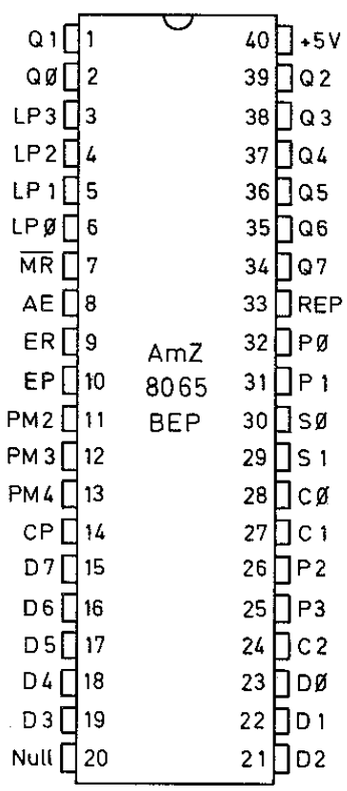


Bild 10.10

der Auswahl des Prüfmodus (C0 . . . C2) verarbeitet, z.B. Korrektur Normal C0=Lo, C1=C2=Hi. Bei der Normal-Korrektur werden die Syndrom-Bits (Bitmuster) laufend geteilt, bis ein Fehler entdeckt wurde (EP wird Hi). Generell wird der BEP in **drei Betriebsarten** eingesetzt: **Schreiben** (Daten auf Disk, Berechnung des Prüfbitmusters), **Lesen** (aufspüren von Fehlern beim Auslesen der Disk), **Korrektur** (Syndrom-Bitmuster im Normal-Korrekturmodus, Anzeige an EP und Ausgabe des Fehlermusters an LP0 . . . LP3 und Q0 . . .Q7). Der BEP ist für Datenraten bis zu 20 MBit/s ausgelegt. ausgelegt.

Zuletzt sollen noch einige Typen von Bausteinen genannt sein, die noch künftig die Z 8000-Familie erweitern werden:
AmZ 8052 CRT-Steuerbaustein,
AmZ 8068 DCP-Datenverschlüsselungs-Prozessor,
Z 8090Z-UPC Peripherie-Steuerbaustein.

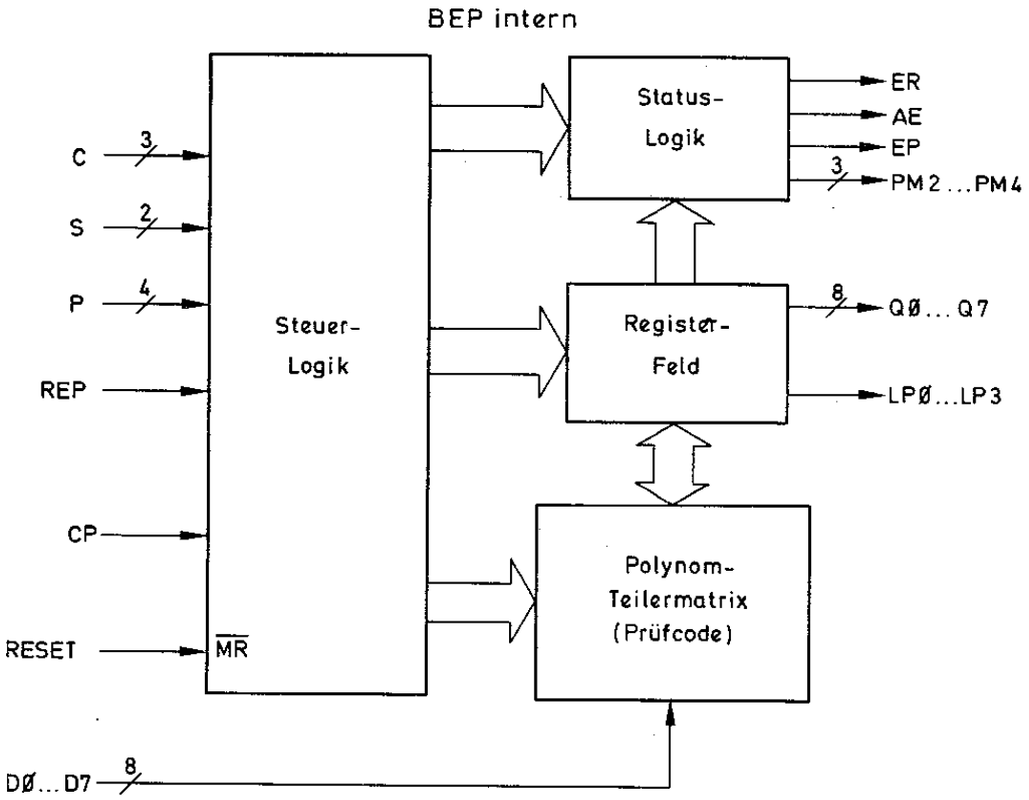


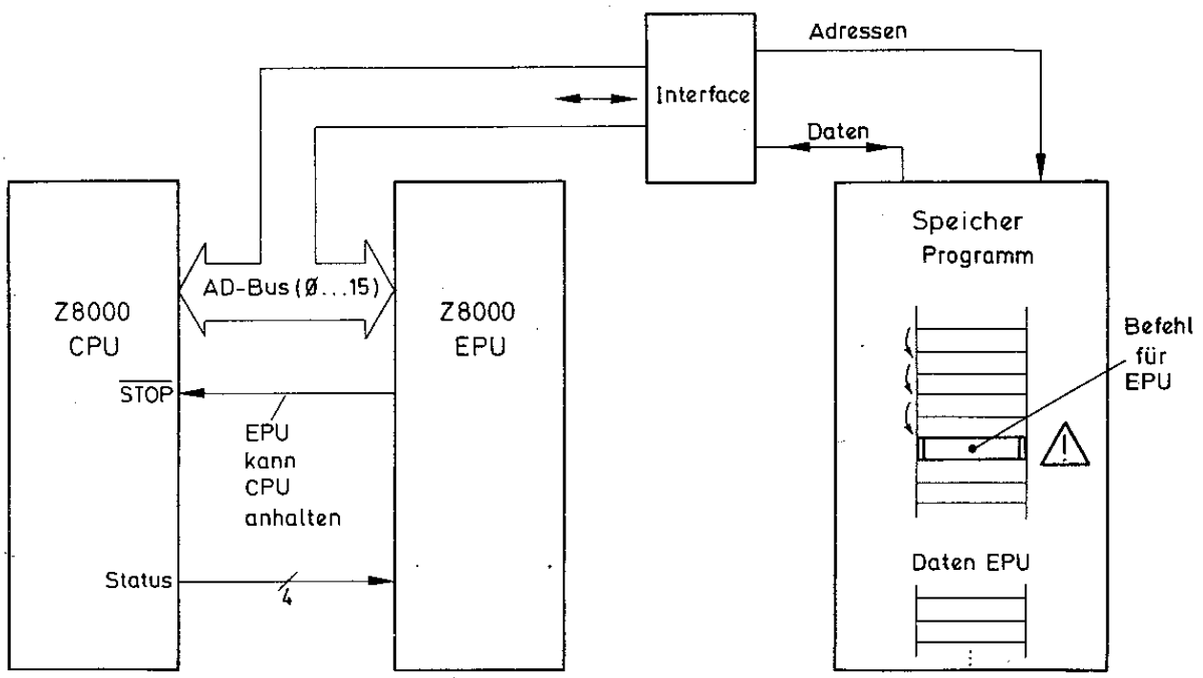
Bild 10.11

10.2 RECHNERKONZEPT MIT ENTLASTUNGSPROZESSOREN, DIE Z 8000-EPU

Die V-Maske der Z 8000-CPU sind durch das EPA (EPU)-Bit im FCW-Register (siehe S. 2-30 und 2-31) mit der Möglichkeit ausgestattet, einen Entlastungsprozessor, genannt EPA (**E**xtended **P**rocessing **A**rchitecture) oder EPU (**E**xtended **P**rocessing **U**nit) aufzunehmen. Diese Z 8000-EPA ist mit einem eigenen Befehlssatz ausgestattet. Der EPA-Befehlssatz beinhaltet sehr komplexe und normalerweise zeitintensive Prozeduren, von denen die CPU entlastet werden soll (z.B. Gleitkommarechnung, Grafiksoftware oder spezielles E/A-Handling). Ein Z 8000-System (eine CPU) erlaubt die Integration bis zu vier EPAs, die diese schwierigen, spezifischen Aufgaben übernehmen können. Die EPAs übertreffen in ihren Disziplinen die CPU bezüglich der Ausführungszeit bei weitem, d.h. ihr interner Zuschnitt und das "high density" MOS-Design erlauben beträchtlich kürzere Ausführungszeiten wie die CPU selbst.

Die prinzipielle Funktion und eine Verknüpfung von CPU mit EPA zeigt **Bild 10.12.** Indem die EPU direkt am AD-Bus der CPU anliegt, kann sie die Befehle prüfen, die die CPU holt. Die EPU erfährt dies über den Bus, den vier CPU-Statusleitungen ST0 . . . ST3 und den üblichen Steuerleitungen, wenn die CPU Befehle aus dem Speicher holt. Tritt an der gezeigten Stelle im Programm ein EPA-Befehl auf, handelt die EPA. Sie kann die CPU über die $\overline{\text{STOP}}$-Leitung anhalten, so daß sie einen Buszugriff starten kann. Folgende EPU-Betriebsarten sind möglich:

- Datentransfer: EPU-Register von/zu Hauptspeicher (2 Befehle)
- Datentransfer: EPU-Register von/zu CPU-Register (2 Befehle)
- Interne EPU-Operationen (1 Befehl)
- Status-Transfer zwischen EPUs und dem FCW-Register der CPU (2 Befehle)



⚠ bei einem CPU-Bef.-Holzyklus. An dieser Stelle bemerkt die EPU einen an sie gerichteten Befehl im Programm.

Blid 10.12

Die Aktivierung der EPU im System ist, wie genannt, davon abhängig, ob das EPU-Bit im FCW-Register der CPU gesetzt ist. Das Anhalten der CPU bzw. die Ausführung von EPU-Operationen von der EPU bedingt, daß das EPU-Bit gesetzt sein muß. Ist es nicht gesetzt und erscheint im Programmverlauf eine Anweisung für die EPU, wird diese folgendermaßen gehandhabt. In Bild 2.22 auf Seite 2-35 ist die Z 8000-Programm-Sprungtabelle dargestellt. Wird von der CPU ein EPU-Befehl als nicht implementierte Instruktion (Befehl) erkannt, erfolgt in der CPU ein interner Trap (ein Extended Instruction Trap). Dieser bewirkt, daß die CPU auf den Programmzählerwert SSXX (PSAP) 08H gesetzt wird. Der übernommene EPU-Befehl (z.B. 0EXXH) dient dabei als Kennung für die Trap-Serviceroutine, die jetzt die CPU anstelle der EPU ausführt, um den EPU-Befehl auf diese abzuarbeiten. Man kann hier auch sagen, daß die CPU die EPU simuliert, wenn auch langsam. Die CPU kommt jedoch nach der Ausführung zum gleichen Ziel wie die EPU. Der Vorteil der Methode liegt darin, daß ein bestimmtes Anwenderprogramm mit laufenden EPU-Befehlen unmittelbar getestet werden kann, ohne daß eine EPU im System zur Verfügung steht. Ist die EPU dann in das System integriert, braucht nur das EPU-Bit im FCW-Register gleich 1 gesetzt werden. Die Traproutine kann man dann vergessen. Insgesamt sind es sieben EPA-Befehle, die die EPU in Korrespondenz mit der CPU-Hardware ausführen kann. Diese Befehle können in jeweilig vier (max.) Adressierungsarten gewählt werden. Das Befehlsformat der EPU ist identisch gegliedert wie das der CPU. Sechs Opcodes stehen als EPU-Befehle fest, die in der CPU einen Trap auslösen, wenn sie sie bearbeiten soll:

0E, 0F, 4E, 4F, 8E, 8F –
alle Werte in HEX

In **Bild 10.13** schließlich ist fast ein wenig vorsichtig angedeutet, was uns in den 80er Jahren erwartet. Multiprozessorsysteme dieser Art werden im Verlauf dieses Jahrzehnts einem Industrie-Standard entsprechen. Erinnern wir uns noch einmal an das erste Kapitel: Was

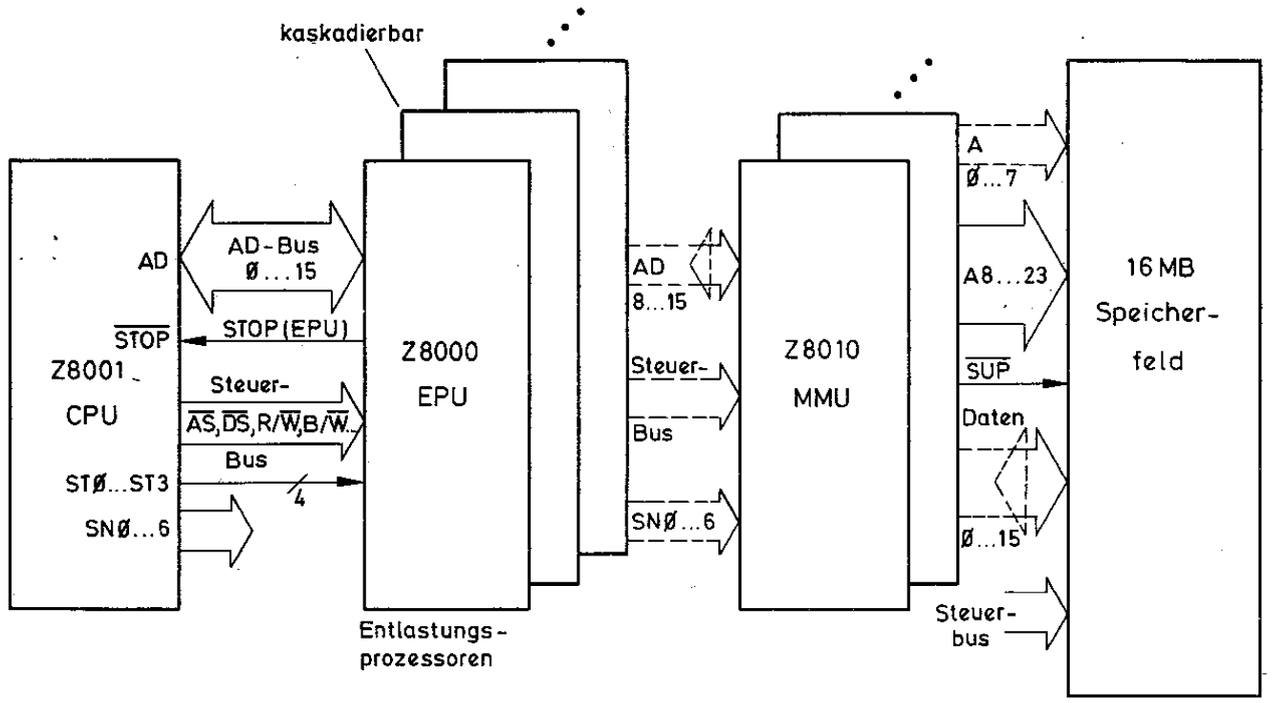


Bild 10.13

ist ein 16-bit-Computer? Die Struktur von Computern nach Bild 10.13 sind auch 16-bit-Systeme. Man sieht ihnen ihre Raffinessen nicht an, weil der Anblick von Platinen nicht viel hergibt. Man wird spätestens dann die Extremität solcher 16-bit-Strukturen begreifen, wenn man mit ihnen aktiv arbeitet. Die Bausteine allein können keine Angst verbreiten. Doch die Aufgaben, die Computer dieses Schlages bewältigen können, werden schwindelerregende Vorstellungen erwecken, — speziell dann, wenn solche kleinen Monster bereits im Einsatz sind.

Fachwörterverzeichnis

access — *Zugriff*
Möglichkeit, eine bestimmte Speicherstelle anzusprechen (zu adressieren).

access time — *Zugriffszeit*
Die zwischen dem Anlegen der Adresse und der Abgabe von Daten eines Speichers erforderliche Zeit.

accumulator — *Akkumulator*
Register, in dem die Ergebnisse arithmetischer, logischer und E/A-Operationen gebildet werden. Der Inhalt des A. kann gelöscht, geprüft, komplementiert, inkrementiert, dekrementiert oder nach links oder rechts verschoben werden.

acknowledge — *Quittierung*
Bestätigung für ein empfangenes Signal.

ADC (Analog Digital Converter) — *Analog/Digital-Umsetzer, A/D-Wandler*
Schaltung zur Umwandlung eines in analoger Form vorliegenden Signales in den äquivalenten Digitalwert.

adder — *Addierer, Addierwerk*
Dient zur Bildung der Summe von zwei oder mehreren Eingangssignalen.

address — *Adresse, adressieren*
Ein Wort zur eindeutigen Identifizierung eines Speicherplatzes oder anderer Datenquellen und Senken.

algorithm — *Algorithmus*
Eine Rechenvorschrift zur Lösung einer Aufgabe in einer Reihe von Schritten. Jedes Computerprogramm stellt im Grunde einen Algorithmus dar.

ALGOL (ALGOrithmic Language) —
Problemorientierte, höhere Programmiersprache für technisch-wissenschaftliche Programme.

alphanumeric — *alphanumerisch*
Gemischte Darstellung aus Ziffern und Buchstaben.

ALU (Arithmetic and Logical Unit) — *Arithmetisch-logische Einheit, Rechenwerk*
Teil der Computer-Zentraleinheit. In ihr werden die arithmetischen und logischen Operationen ausgeführt. Diese Operationen werden durch Befehle in binär codierter Form ausgeführt.

ASCII (American Standard Code for Information Interchange) — *Amerikanischer Code für Informationsaustausch*
Alphanumerischer Standardcode meist „Aski" gesprochen.

assembler — *Assembler oder Assemblierer*
Ein Programm, das Programme aus der Assemblersprache (assembly language) in Maschinensprache übersetzt. Formale Fehler werden von diesem Programm erkannt

assembly language — *Assemblersprache*
Programmiersprache, bei der anstelle des binären Maschinencodes einfach zu merkende Kürzel (Mnemoniks) verwendet werden. Außer den absoluten Adressen können symbolische Adressen verwendet werden. Anweisungen an den Assembler können mit Pseudo-Befehlen ausgeführt werden.

asynchronous — *asynchron*
Taktunabhängig, ohne Takt, nicht synchronisiert.

background program — *Hauptprogramm*
Hauptprogramm in einem Computersystem, das imstande ist, Unterbrechungen zu verarbeiten.

BASIC (Beginners All purpose Symbolic Instruction Code)
Einfach zu lernende höhere Programmiersprache.

baud rate — *Baud-Rate, Baud-Zahl*
Einheit der Übertragungsgeschwindigkeit. Ist gleich der Anzahl der Signalvorgänge pro Sekunde.

BCD (Binary Coded Decimal) — *Binär codierte Dezimalzahl*
Ein Code, bei dem jede Ziffer einer Dezimalzahl in ein binäres 4-Bit-Wort umgewandelt wird. Auch 1-2-4-8-BCD-Code genannt.

benchmark program — *Bewertungsprogramm*
Ein Programm zur Bewertung der Leistungsfähigkeit eines Computers.

bidirectional — *bidirektional*
In beide Richtungen arbeitend.

bit (binary digit) — *Bit*
Binäre Informationseinheit. Binärzeichen oder Dualziffer.

Boolean algebra — *Boole'sche Algebra*
System von Rechenregeln für binäre Variable.

branch — *Verzweigung*
Die Fortsetzung eines Programms an einer von zwei verschiedenen Adressen.

branch instruction — *Verzweigungsbefehl*
Befehl zur Weiterführung eines Programms an einer von zwei möglichen Adressen, abhängig von einer Bedingung.

breakpoint — *Haltepunkt*
Um einem Anwender Prüf- oder Eingabemöglichkeiten zu geben, kann ein Programm vor der Ausführung eines an einem bestimmten Haltepunktes gespeicherten Befehls gestoppt werden.

buffer — *Puffer*
Kleiner Speicher, in dem Daten kurzzeitig zwischengespeichert werden.

bug — *Fehler (wörtl. Wanze)*
Fehler in einem Software-Programm.

bus — *Bus, Sammelschiene*
Mehrfach-Datenleitung, an der verschiedene Einheiten gleichzeitig angeschlossen sind (Adressenbus, Datenbus, Steuerbus).

BUSEN (BUS ENable) — *Freigabesignal für den Bus*

byte — *Byte*
Bezeichnung für eine Gruppe von 8 Bits, die gemeinsam verarbeitet werden.

card puncher — *Lochkartenstanzer*

card reader — *Lochkartenleser*

carriage return — *Wagenrücklauf*

carry — *Übertrag*
Ein Signal oder ein Ausdruck, der auftreten kann, wenn die Summe zweier Ziffern mit demselben Stellenwert gleich oder größer als die Basis des verwendeten Zahlensystems ist.

carry look ahead — *Parallelübertrag*
Zur Erzielung einer raschen Verarbeitung eines Übertrages entsprechende logische Verknüpfungen.

cascade — *Kaskade, kaskadieren*
Beliebige Erweiterung der zu verarbeitenden Wortlänge durch Serienschaltung mehrerer Prozessor-Bausteine (slices).

character — *Zeichen, Symbol*
Kleinste Einheit, aus der sich Daten zusammensetzen. (Binäre Zeichen, Buchstaben, Ziffern und Sonderzeichen.)

chip — *Chip*
Monolithisches Halbleiterkristall-Plättchen oder Baustein, dessen Schaltung auf einem Halbleiterkristall aufgebracht ist.

chip-enable — *Baustein-Freigabe, Chip-Freigabe*
Freigabe eines Bausteines über ein Baustein-Freigabesignal.

chip-select — *Baustein-Auswahl, Chip-Auswahl*
Auswahl eines Bausteines über ein Baustein-Auswahlsignal.

chip slices — *(wörtl.) Chip-Scheiben*
Prozessorelemente für 2 oder 4 Bits, die zum Aufbau eines Mikroprozessors mit beliebiger Wortlänge zusammengeschaltet werden können.

clear — *löschen*
Zurücksetzen auf Null.

clock — *Takt*
Takt oder Taktimpuls. Bestimmt die Ausführungsdauer der Befehle.

compare — *vergleichen*

compatibel — *kompatibel, verträglich*
Software-kompatibel sind zwei Rechner, deren Programme ohne Änderung untereinander austauschbar sind.
Hardware-kompatibel sind Einheiten, die ohne Anpaßschaltung (z.B. bezüglich Signalpegel) zusammengeschlossen werden können.

compiler — *Compiler, Kompilierer*
Übersetzungsprogramm, das Programme aus einer höheren Programmiersprache in Maschinensprache umsetzt.

conditional — *bedingt*
Ausführung einer Operation entsprechend des Vorhandenseins bestimmter Bedingungen.

conditional jump — *bedingter Sprung*
Ein Sprung, der nur bei Vorliegen bestimmter Bedingungen ausgeführt wird.

control unit — *Steuereinheit, Kontrolleinheit*
1. Steuereinheit für die Reihenfolge der Befehle eines Maschinenprogramms.
2. Steuereinheit für periphere Geräte.

CPU (Central Processing Unit) — *Zentraleinheit*
Zentraleinheit oder Teil der Zentraleinheit. Wird manchmal auch als Mikroprozessor bezeichnet und besteht aus der arithmetisch-logischen Einheit (ALU), Kontrolleinheit (CU) und Registern.

CRC (Cyclic Redundancy Check) — *Zyklische Blockprüfung*
Verfahren zur Feststellung von Fehlern bei der Übertragung und Speicherung von Datenblöcken.

CRT (Cathode Ray Tube) — *Kathodenstrahlröhre*
Kathodenstrahlröhre für Datensichtgeräte.

cross assembler —
Ein Assembliererprogramm, das auf dem Computer 1 läuft und ein in Assemblersprache des Computers 2 geschriebenes Programm in die Maschinensprache von 2 übersetzt.

cycle — *Zyklus*
Ein Zeitintervall, in dem bestimmte Vorgänge ablaufen und abgeschlossen werden.

DAC (Digital to Analog Converter) — *Digital/Analog-Umsetzer, D/A-Wandler*

data bus — *Datenbus*
Gemeinsame Datenleitung, an die mehrere Einheiten gleichzeitig angeschlossen werden können.

debug, to — *Fehlersuchen (wörtl. Entwanzen)*
Suchen und Beseitigen von Fehlern in der Hardware und insbesondere in der Software.

decimal adjust — *Dezimalkorrektur, Dezimalanordnung*
Bei der Dezimalkorrektur wird der Inhalt eines Registers entnommen und zur Schaffung des dezimalen Äquivalents der Binärzahl neu angeordnet.

decrement — *dekrementieren*
Schrittweises Vermindern um Eins.

development system — *Entwicklungssystem*
Mikrocomputer-Systeme zum raschen Testen von Software und Hardware, sowie dem Zusammenarbeiten beider. Wird auch zur PROM-Programmierung verwendet.

device — *Baustein, Einheit, Gerät*

diagnotistic routines — *Diagnose-Programme*
Programme zur Aufspürung und Lokalisierung von Fehlern in Hardware und Software.

digit — *Ziffer, Stelle*
Kleinste Einheit in einem digitalen System. Kann im Dezimalsystem die Werte der zehn Dezimalziffern annehmen, in einem Binärsystem die Ziffern Null und Eins. Im Engl. bedeutet digit auch eine Stelle in einer Zahl.

DIP (Dual-In-line-Package) —
Schaltungsgehäuse mit in zwei Reihen angeordneten Anschlüssen.

direct addressing — *direkte Adressierung*
Bei der direkten Adressierung ist die Adresse des Operanden Bestandteil des Befehls.

disk memory — *Plattenspeicher*
Speicher mit großer Kapazität (10^7- 10^9 Bits) aus rotierenden Platten mit magnetisierbarer Schicht. Der Zugriff zu den Speicherstellen ist wahlfrei.

display — *Anzeige*
Nichtschreibende, optische Anzeige von Daten. Reicht von der einfachen Anzeigelampe bis zum Sichtgerät mit Kathodenstrahlröhre (CRT).

DMA (Direct Memory Access) — *direkter Speicherzugriff*
Verfahren zur direkten Übertragung von Daten zwischen Speicher und E/A-Bausteinen, ohne daß die Daten über die CPU laufen müssen.

driver — *Treiber*
Schaltung zur Erzeugung der für das Ansteuern entsprechender Bausteine erforderlichen Leistung.

dynamic RAM (dynamic Random-Access Memory) — *dynamischer Schreib-/Lese-Speicher*
Speicher, in dem die Aufbewahrung der Informationen in elektrischen Ladungen erfolgt. Die Ladungen müssen ständig aufgefrischt werden (ca. alle 1 - 2 ms).

EAROM (Electrically Alterable ROM) — *elektrisch änderbarer Festwertspeicher*
Festwertspeicher, dessen Inhalt sich elektrisch verändern läßt.

editor — *Editor*
Ein Hilfsprogramm zur Unterstützung des Eingebens, Ausgebens, Korrektur und Speicherung von Programmen.

effective address — *effektive Adresse*
Bezeichnung für die absolute Adresse bei indirekter und indizierter Adressierung.

emulation — *Emulation*
Softwaremäßige Nachbildung (Simulation) eines Computers, wobei nur das äußere Verhalten eines Systems nachgebildet wird.

enable signal — *Freigabe-Signal*

encode — *codieren*
Das Zuordnen von Zeichen aus einem Zeichenvorrat zu Zeichen aus einem anderen Zeichenvorrat.

EPROM (Erasable-Programmable ROM) — *lösch- und programmierbarer Festwertspeicher*
Festwertspeicher, dessen gesamter Inhalt sich durch Bestrahlung mit UV-Licht löschen und sich anschließend wieder neu programmieren läßt.

erase — *löschen*
Löschen des Inhalts eines Speichers, z.B. bei einem EPROM mittels UV-Licht.

even parity — *gerade Parität*
Anzahl der auf "1" befindlichen Bits (pro Wort oder Byte) einschließlich des Paritätsbits wird gerade gemacht.

exchange, to — *austauschen*
Austauschen des Inhalts von Speicherplätzen mit Registern, Akkumulator etc.

execution time — *Befehls-Ausführungszeit*
Die Zeit, die zur vollständigen Ausführung eines Befehls erforderlich ist.

fetch, to — *holen, herausholen, abrufen*
Abrufen eines oder mehrerer Bytes (Befehle oder Daten) aus dem Speicher. Geschieht im Abrufzyklus (fetch machine cycle).

file — *Datei*
Zusammenstellung von Daten für einen bestimmten Zweck. Mehrere Sätze bilden eine Datei, mehrere Dateien eine Datenbank.

firmware — *Firmware*
In Festwertspeichern aufbewahrte Software.

flag — *Flagge, Kennzeichen*
Das Flag-Bit wird von der Software gesetzt, um einen bestimmten Zustand anzuzeigen und festzuhalten. Kann bei Bedarf später abgefragt werden, z.B. für bedingte Verzweigungen oder Sprünge.

floate, to — *floaten*
Das Versetzen eines Anschlusses oder einer Leitung in den hochohmigen Zustand. Beim DMA z.B. bringt sich die CPU in diesen Zustand und trennt sich damit vom angeschlossenen Bus.

floppy disk — *Magnetplatten-Speicher*
Preiswerte magnetische Speicherplatten mit wahlfreiem Zugriff und großer Speicherkapazität.

flow-chart — *Flußdiagramm, Ablaufdiagramm*
Dient zur Darstellung des Verarbeitungsablaufes der Daten in Diagrammform. Für komplexe Programme ein unentbehrliches Hilfsmittel.

FORTRAN (FORmula TRANslation) —
Problemorientierte Programmiersprache für technische und wissenschaftliche Programme.

FPLA (Field Programmable Logic Array)
— Vom Anwender programmierbare logische Anordnung.

framing — *Rahmung, Einrahmung*
Bei asynchroner Datenübertragung gehen jedem Zeichen ein oder mehrere Startbits voraus und werden von einem oder mehreren Stopbits abgeschlossen.
Das Zeichen ist somit von Start- und Stopbits "eingerahmt".

hand shaking — *Quittungsbetrieb*
Geräte mit unterschiedlichen Reaktionsgeschwindigkeiten verständigen sich über den Beginn und Ende eines Austausches von Daten.

hardware — *Hardware*
Gemeinsamer Begriff für Bauteile und Geräte.

high — *hoch*
Zustand mit hohem Pegel (logisch H). Bei positiver Speisespannung entspricht "H" dem Zustand "Ein" (DIN 41785).

high-level language — *höhere Programmiersprache*
Sammelbegriff für alle problemorientierten Programmiersprachen (z.B. FORTRAN, BASIC, ALGOL, PL/1). Im Gegensatz hierzu stehen die maschinenorientierten Programmiersprachen (Assemblersprachen).

immediate addressing — *unmittelbare Adressierung*
Bei der unmittelbaren Adressierung enthält der Befehl keine Adresse sondern den Operanden direkt. Da ein Befehl nur Konstante beinhalten kann, ist dies nicht für Variable möglich.

increment — *inkrementieren*
Schrittweises Erhöhen um 1, z.B. automatisches Inkrementieren des Programmzählers zur Bildung der neuen Adresse.

index register — *Index-Register*
Register, die zur Änderung von Adressen verwendet werden, die in Befehlen enthalten sind (indirekte Adressierung).

indexed addressing — *indizierte Adressierung*
Bei der indizierten Adressierung wird der im Befehl angegebenen Adresse der Inhalt eines Indexregisters hinzu addiert, um die Adresse des Operanden zu erhalten. Durch Änderung des Inhalts des Indexregisters kann die effektive Adresse abgeändert werden, ohne daß der Befehl modifiziert werden muß. Besonders nützlich bei Schleifen für die Abarbeitung von Tabellen.

indirect addressing — *indirekte Adressierung*
Bei der indirekten Adressierung ist der im Befehl angegebene Operand die Adresse des Operanden.

inhibit — *Sperr . . .*

instruction — *Befehl*
Anweisung für den Computer zur Durchführung bestimmter Operationen.

instruction set — *Befehlssatz, Befehlsvorrat*
Der Satz von Befehlen in Maschinensprache, der von einem Rechner ausgeführt werden kann. Wichtige Befehlsarten sind:
1. Arithmetische und logische Befehle.
2. Befehle zur Steuerung des Programmablaufs.
3. Befehle zum Transfer von Daten.

Interface — *Interface, Schnittstelle, Anpassungsschaltung*
Schnitt- oder Trennstelle zwischen zwei Geräten, pegel- und ablaufmäßig genormt. Elektronische Schaltung, die zwei Geräte oder Bausteine einander anpaßt.

Interrupt — *Unterbrechung, Programmunterbrechung*
Das momentan ablaufende Programm wird unterbrochen, alle wichtigen Registerinhalte werden gerettet (saved) und ein Unterbrechungs-Serviceprogramm ausgeführt. Danach werden die geretteten Werte wieder in die Register geladen und das Hauptprogramm wird fortgesetzt. Dient zur Behandlung von asynchronen (insbesonders E/A) Vorgängen.

interrupt request — *Unterbrechungs-Anforderung*
Signal von einer externen Einheit, die eine Unterbrechung anfordert.

interrupt service routine — *Unterbrechungs-Serviceprogramm*
Unterprogramm, zu dem bei Eintreffen einer Unterbrechungs-Anforderung gesprungen wird.

I/O (Input/Output) — *E/A (Eingabe/Ausgabe)*

I/O-port — *Eingabe/Ausgabe-Kanal*

jump — *Sprung*
Fortsetzung des Programms mit einem anderen als dem nächstfolgenden Befehl.

keyboard — *Tastatur*

label — *Marke, Markierung, Kennzeichen*
In Programmiersprachen eine symbolische Adresse.

latch — *Signalspeicher*
Signalspeicher oder Auffangspeicher, meist ein Flip-Flop.

least significant bit (LSB) — *Bit mit dem niedrigsten Stellenwert*

LIFO (Last-In/First-Out) —
Speicherprinzip, bei dem die zuletzt gespeicherten Daten wieder zuerst ausgegeben werden. Entnahmeprinzip beim Stapelspeicher (stack).

linking loader — *Binde-Lader*
Ein Ladeprogramm zum Zusammenfügen mehrerer unabhängig voneinander erstellten Programmteile zu einem Gesamtprogramm.

listing — *Auflistung, gedruckte Liste*

literal — *Literal*
Wenn eine Konstante, die als Rechengröße benötigt wird und sich während des Programmablaufs nicht ändert, direkt als Operand angegeben wird, nennt man sie Literal.

loader — *Lader*
Programm, das ein Maschinenprogramm an eine bestimmte Stelle im Speicher lädt. Hierbei erfolgt u.U. eine Umrechnung von Adressen.

location — *Speicherplatz, Speicherzelle*
Speicherstelle eines Wortes, die durch eine Adresse eindeutig gekennzeichnet ist.

loop — *(Programm-)Schleife*
Eine Serie von Befehlen, die zur Lösung einer Aufgabe mehrmals nacheinander durchlaufen werden kann.

LSI (Large Scale Integration) — *Hoher Integrationsgrad*
Komplexe Schaltungen mit mehr als 1000 Gattern auf einem einzelnen Chip.

machine language — *Maschinensprache*
Die Maschinensprache besteht aus Befehls-Bitmustern, die der Computer direkt decodieren und ausführen kann. Programme in anderen Sprachen müssen in Maschinensprachen übersetzt werden.

machine program — *Maschinenprogramm*
Programm in Maschinensprache.

macro-instruction — *Makrobefehl*
Eine Folge von Mikrobefehlen, die in einem Makrobefehl zusammengefaßt werden.

mask — *Maske*
Ein Bitmuster zur gezielten Ausblendung von Binärstellen.

mass storage — *Massenspeicher*
Speicher mit sehr großer Speicherkapazität, z.B.: Magnettrommeln, Magnetplatten — im Gegensatz zum Arbeits- oder Programmspeicher.

memory — *Speicher, Speicherbaustein*
Einrichtung zur Aufbewahrung von Informationen (Daten, Befehle, Adressen etc.) über einen beliebig langen Zeitraum.

microcomputer — *Mikrocomputer*
Ein Computer mit einem Mikroprozessor als Zentraleinheit, sowie Programmspeicher (meist ROM oder PROM), Datenspeicher (RAM) und E/A-Bausteinen.

micro instruction — *Mikrobefehl*
Computer führen Befehle in Maschinensprache meist als Sequenz (Folge) kleinerer Programmschritte in Form von Mikrobefehlen aus.

microprocessor — *Mikroprozessor*
Zentraleinheit (CPU = Central Processing Unit) auf einem oder mehreren Chips.

microprogram — *Mikroprogramm*
Folge von Mikrobefehlen.

mnemonic — *Mnemonik*
Mnemonischer oder mnemotechnischer Code. Ein Code, dessen Abkürzungen ein Hinweis auf dessen Eigenschaften zur Gedächtnisstütze enthält. Es handelt sich insbesonders um leicht zu merkende, alphanumerische Kürzel für Befehle, die wesentlich leichter als ein Zahlencode zu behalten sind.

modem (MOdulator/DEModulator) — *Modem*
Kombinierter Modulator und Demodulator für die Übertragung von Daten.

monitor — *Monitor*
Ein Programm zur Steuerung oder Koordination anderer Programmteile oder selbständiger Programme.

most significant bit (MSB) — *Bit mit dem höchsten Stellenwert*

MSI (Medium Scale Integration) — *Mittlerer Integrationsgrad*
Schaltungen mit weniger als 100 Gattern auf einem einzelnen Chip.

multibyte instruction — *Mehrwort-Befehl*
Ein Befehl, dessen erforderliche Bit-Zahl größer als die Wortlänge des Computers ist. Dieser Befehl muß daher in zwei oder mehr Speicherplätze abgelegt werden.

multilevel interrupt — *Mehr-Pegel-Unterbrechung*
Eine Unterbrechung mit hoher Priorität kann eine andere mit niedrigerer Priorität unterbrechen.

multiplex — *Multiplex*
Im allgemeinen werden im Zeitmultiplex-Verfahren die auf einer größeren Anzahl von Kanälen vorliegenden Daten auf eine geringere Anzahl (oder einen einzelnen) Kanal umgesetzt.

nesting — *Verschachtelung*
Wenn von einem Unterprogramm ein weiteres Unterprogramm aufgerufen wird (z.B. bei Unterbrechungs-Programmen).

number representation — *Zahlendarstellung*
Im Rechner werden Zahlen im Dualsystem oder im BCD-Code verarbeitet. Zur Eingabe oder Ausgabe werden drei oder vier Bits im Oktal- und Hexadezimalsystem zusammengefaßt.

object code — *Maschinencode*
Anweisungen, die von der Maschine unmittelbar interpretiert werden können.

object program — *Maschinenprogramm*
Ein Programm in Maschinensprache, das vom Assembler oder Kompilierer erzeugt wurde und direkt geladen werden kann.

odd parity — *ungerade Parität*
Anzahl der auf "1" befindlichen Bits (pro Wort oder Byte) einschließlich des Paritätsbits wird ungerade gemacht.

on-line — *(wörtl.) "in der Leitung"*
Mit der Anlage verbundener, synchronisierter Betrieb.

operand — *Operand*
Information oder Rechengröße, an der eine entsprechende Operation durchgeführt wird.

overflow — *Überlauf*
Überschreiten der Stellenanzahl durch das Ergebnis.

PACE (Processing And Control Element) — *Mikroprozessor von National Semiconductor*

paged addressing — *Seiten-Adressierung*
Durch Aufteilung des Speichers in Abschnitte (Seiten) kann man den Aufwand der Adressierung innerhalb der gleichen Seite verringern, indem kürzere Adressen verwendet werden. Der Nachteil ist, daß eine Adressierung außerhalb der Seite nur mit erhöhtem Aufwand möglich ist (Verbesserung siehe "relative addressing").

parity — *Parität*
Die Kontrolle der Parität ist ein Hilfsmittel zur Datensicherung. Durch Hinzufügen eines Paritätsbits wird die Anzahl der "1"-Bits gerade (even) oder ungerade (odd) gemacht und bei Empfang kontrolliert.

peripheral (device) — *Peripheriegerät, peripheres Gerät*
Alle Speicher- und Eingabe/Ausgabe-Einheiten, die nicht zur Zentraleinheit gehören und vom Computer angesprochen werden können.

PLA (Programmable Logic Array) — *programmierbare logische Anordnung*
Integrierte Schaltung, mit der sich beliebige Verknüpfungen binärer Eingangsgrößen durchführen lassen (z.B. Code-Wandlung).

PL/M (Programming Language for Microcomputer) — *Programmiersprache für Mikrocomputer*
Basierend auf PL/1 (Programming Language 1 = höhere Programmiersprache).

PMOS — *P-Kanal-MOS*
MOS-Technologie. Besitzt gegenüber NMOS größere Kristallflächen- und Leistungsbedarf sowie höhere Schaltgeschwindigkeiten.

pointer — *Zeiger, Adressenverweis*
Ein Speicherplatz, der eine Adresse enthält. Bei häufiger Verwendung wird hierfür ein Register benützt. (z.B. instruction counter, stack pointer).

pooling — *Pooling*
Bildung von Pools. Das Abspeichern von aus mehreren Bytes bestehenden Befehl-Codes (multibyte instruction object codes) in dafür bestimmte Speicher, von dort können sie mit speziellen Ein-Byte-Referenzen abgerufen werden. Diente ursprünglich zur Einsparung von Speicherraum, kann jedoch zur direkten Steuerung der Mikrocomputer-Logik durch externe Logik verwendet werden.

pop — *pop*
Entnehmen von Informationen aus dem Stapelspeicher.

port — *Kanal*
Eingabe- oder Ausgabe-Baustein.

priority — *Priorität, Rangfolge*
Teile eines Systems mit höherer Priorität werden bei Konkurrenzfällen mit Vorrang behandelt.

program — *Programm*
Folge (oder Sequenz) von Befehlen zur Lösung einer Aufgabe.

program counter — *Programmzähler*
Register, das die Adresse des nächsten auszuführenden Befehls enthält.

PROM (Programmable ROM) — *programmierbarer Festwertspeicher*
Vom Benutzer programmierbarer Speicher, dessen Inhalt jedoch nicht mehr gelöscht werden kann.

pseudo instruction — *Pseudobefehl*
Eine Anweisung an den Assembler, die nicht übersetzt wird.

push — *push*
Einspeichern von Informationen in den Stapelspeicher.

RAM (Random Access Memory) — *Speicher mit wahlfreiem Zugriff*
Schreib-/Lese-Speicher mit wahlfreiem Zugriff.

random access — *wahlfreier Zugriff*
Bei Speichern mit wahlfreiem Zugriff ist die Zugriffszeit unabhängig von der gewählten Adresse (im Gegensatz zu den sequentiellen Speichern).

real-time — *Realzeit, Echtzeit*
Ein Computer, der im Echtzeit-Betrieb arbeitet, nimmt direkt Einfluß auf den Prozeß in den er einbezogen ist.

redundancy — *Redundanz*
Die zur Darstellung einer Information zur Verfügung stehenden, aber nicht benötigten Zeichen werden als Redundanz (wörtl. Weitschweifigkeit) dieser Information bezeichnet. Je höher die Redundanz, desto sicherer ist der Code gegen Fehler, er wird jedoch umso umfangreicher.

register — *Register*
Eine Anordnung die imstande ist, kleine Einheiten digitaler Informationen vorübergehend zu speichern und mit kurzer Zugriffszeit abzugeben. Auf die Bits wird meist parallel zugegriffen.

register-register instruction — *Register-Register-Befehl*
Ein Befehl, bei dem sich der Operand in einem Register befindet und das Ergebnis wieder in einem Register abgelegt wird.

relative addressing — *relative Adressierung*
Verbesserte Form der Seiten-Adressierung. Es wird eine Abweichung von der vorherigen Adresse von ± 1/2 Seite angegeben. Dadurch wird der erhöhte Aufwand bei den Seitenübergängen vermieden, der jedoch bei Adressierungen außerhalb des relativen Seitenbereiches wieder voll auftritt.

relocatable — *verschiebbar*
Nicht an eine absolute Adresse gebunden.

relocating loader — *Lader für verschiebbare Programme*
Von diesem Lader kann ein verschiebbares Objektprogramm an einen beliebigen Platz im Programmspeicher für die Ausführung gebracht werden.

REPROM (REprogrammable PROM) — *Wiederprogrammierbares ROM*
Ein Festwertspeicher, der gelöscht und wieder programmiert werden kann.

resident — *(wörtl.) ortsansässig*
a) Auf dem eigenen Computer laufend (Gegenteil: cross)
b) Ständig im Hauptspeicher befindlich.

restart — *neuerlicher Start*
Neuer Start nach einer Unterbrechung.

return address — *Rückkehradresse, Rücksprungadresse*
Die Adresse, zu der nach Beendigung des Unterprogramms zurückgekehrt wird.

ROM (Read Only Memory) — *Festwertspeicher, Nur-Lese-Speicher*
Festwertspeicher mit konstantem unveränderlichem Inhalt, z.B. Programm, Konstante.

scratch pad — *Zwischenregister (wörtl. "Notizblock")*
Ein Register, das für kurzfristige Aufbewahrung von Zwischenergebnissen verwendet wird.

sequential access — *sequentieller Zugriff*
Der Zugriff zu gespeicherten Daten ist nur nach dem Lesen aller vorher gespeicherten Informationen möglich, z.B. bei einem Magnetband.

shift — *Verschiebung*
Eine Operation, bei der eine Anzahl von Bits eine oder mehrere Stellen nach links oder rechts verschoben wird.

sign — *Vorzeichen*

simulation — *Simulation*
Anstelle eines tatsächlichen Systems wird ein Modell dieses Systems verwendet, das in seinem äußeren Verhalten der internen Arbeitsweise dem wirklichen System gleicht.

simulator — *Simulator*
Ein Programm zur Nachbildung von Befehlen. Wird meist zum Überprüfen eines fertigen Programms verwendet.

slice — *(wörtl.) Scheibe*
Prozessorelement (für 2 oder 4 Bits). Mehrere hiervon können zum Aufbau eines Mikroprozessors mit beliebiger Wortlänge zusammengeschaltet werden (bit slice).

software — *Software*
Zur praktischen Verwendung der Computer-Hardware benötigt man Programme, die als Software bezeichnet werden. System-Software sind alle Grundprogramme, Assembler, Kompilierer, Lader etc. Anwender-Software sind alle Programme zur Lösung spezieller Anwenderprobleme.

source program — *Quellenprogramm*
Ein Programm in einer Programmiersprache. Dieses muß anschließend assembliert oder kompiliert werden.

stack — *Stapelspeicher, Kellerspeicher*
Ein Speicherbereich außerhalb des Steuerwerks, aus dem Informationen nur am gleichen Speicherplatz entnommen werden können, an dem sie eingegeben wurden. (Das Prinzip wird daher auch LIFO = last in first out genannt.)

status — *Zustand*

stack pointer — *Stapelzeiger*
Verweist auf die Adresse des zugänglichen Speicherplatzes des Stapelspeichers, an dem zugefügt oder weggenommen wird.

storage — *Speicher*
Synonym mit memory.

subroutine — *Unterprogramm*
Eine Folge von Befehlen zur Lösung einer Teilaufgabe. Wird vom Hauptprogramm oder anderen Unterprogrammen aufgerufen. Der Abschluß mit einem RETURN-Befehl sichert die Rückkehr zum rufenden Programm.

symbolic address — *symbolische Adresse*
Frei gewählte Bezeichnung eines Speicherplatzes, der vom Assemblierer (oder Kompilierer) in eine absolute Adresse umgesetzt wird.

synchron — *synchron*
Mit festem Grundtakt arbeitend.

tape — *Band*
a) Lochstreifen (paper tape)
b) Magnetband (magnetic tape)

terminal — *Terminal, Datenstation*
Datenstationen für den Benützer, bestehend aus einer Ausgabeeinheit (z.B. Bildschirm, Drucker) und einer Eingabetastatur.

throughput — *Durchsatz*
Die Anzahl der von einem Computer je Zeiteinheit erledigten Aufgaben.

Time-sharing system — *Teilnehmer-System*
Eine Betriebsart, bei der mehrere Benutzer einen Rechner (scheinbar gleichzeitig) für verschiedene Probleme benutzen können.

TTY (Tele TYpewriter) — *Fernschreiber*
Für Mikrocomputer wichtiges E/A-Gerät.

twos-complement — *Zweierkomplement*
Wird aus dem Einerkomplement einer Binärzahl durch Addition von 1 gebildet, z.B. Binärzahl 0100, Einerkomplement 1011, Zweierkomplement 1100. Stellt das binäre Äquivalent eines Zehnerkomplements dar.

UART (Universal Asynchronous Receiver/Transmitter) — *Universeller asynchroner Empfänger/Sender*
Ein Baustein, in dem der Sender die Parallel-Serien-Umsetzung der asynchron zu übertragenden Daten und der Empfänger die entsprechende Serien-Parallel-Umsetzung durchführt.

unconditional jump — *unbedingter Sprung*
Ein Sprung, dessen Ausführung an keine Bedingung gebunden ist.

user program — *Anwenderprogramm*
Spezialprogramm zur Lösung einer Aufgabe des Anwenders. (Gegensatz: Universal- oder Betriebsprogramm).

utility program — *Dienstprogramm*
Allgemeines Hilfsprogramm, das den Umgang mit einem Datenverarbeitungs-System erleichtert und vereinfacht.

vektor interrupt — *gerichtete Unterbrechung*
Unterbrechung, bei der jeder anfordernde Baustein ein eigenes Serviceprogramm erhält.

volatile — *flüchtig*
Speicher, deren Inhalt bei Ausfall oder Abschalten der Stromversorgung verlorengeht, nennt man flüchtig.

word size — *Wortlänge*
Entsprechend dem Anwendungsgebiet werden mehrere Bits zu Worten unterschiedlicher Länge (4, 8, 12, 16 und 32 Bits) zusammengefaßt. Ein Wort mit 8 Bits wird als Byte bezeichnet.

ISBN 3-921803-07-1

te-wi Verlag, München